WYBRANE ZAGADNIENIA INFORMATYKI TECHNICZNEJ

Eksploracja danych i biometria

pod redakcją naukową Wojciecha Kwedlo

Politechnika Białostocka

WYBRANE ZAGADNIENIA INFORMATYKI TECHNICZNEJ
Eksploracja danych i biometria

pod redakcją naukową
Wojciecha Kwedlo

Politechnika Białostocka
OFICYNA WYDAWNICZA POLITECHNIKI BIAŁOSTOCKIEJ
BIAŁYSTOK 2021

Recenzenci:
dr inż. Marek Bolanowski, PRz
dr inż. Kamil Halbiniak, PC
dr inż. Marcin Klimek, PSW
dr inż. Grzegorz Rubin, PWSIiP
dr inż. Mariusz Rybnik, UwB
dr hab. inż. Paweł Topa, prof. AGH

Redaktor naukowy dyscypliny informatyka techniczna i telekomunikacja:
prof. dr hab. Jarosław Stepaniuk

Redakcja i korekta językowa:
Janina Demianowicz

Skład, grafika i okładka:
Marcin Dominów
Zdjęcie na okładce: torstensimon,
https://pixabay.com/pl/photos/kod-binarny-transfer-danych-otw%c3%b3r-6404957/

© Copyright by Politechnika Białostocka, Białystok 2021

ISBN 978-83-66391-84-0 (eBook)
DOI: 10.24427/978-83-66391-84-0

Publikacja jest udostępniona na licencji
Creative Commons Uznanie autorstwa-Użycie niekomercyjne-Bez utworów zależnych 4.0
(CC BY-NC-ND 4.0).
Pełną treść licencji udostępniono na stronie
creativecommons.org/licenses/by-nc-nd/4.0/legalcode.pl.
Publikacja jest dostępna w Internecie na stronie Oficyny Wydawniczej PB.

Oficyna Wydawnicza Politechniki Białostockiej
ul. Wiejska 45C, 15-351 Białystok
e-mail: oficyna.wydawnicza@pb.edu.pl
www.pb.edu.pl

Spis treści

Wstęp 5
Rozdział 1. Algorytm k-średnich na wielordzeniowych procesorach CPU oraz akceleratorach GPU 7
1.1. Algorytm k-średnich 8
1.1.1. Algorytm Lloyda 9
1.1.2. Algorytm pierścienia (annulus) 10
1.1.3. Algorytm Yinyang 12
1.1.4. Rozwiązania początkowe 14
1.2. Platforma testowa 14
1.2.1. Architektura procesorów wielordzeniowych na przykładzie procesora AMD Ryzen Threadripper 1950X 15
1.2.2. Architektura akceleratorów GPU na przykładzie architektury Turing 16
1.3. Eksperymentalne porównanie algorytmów 18
Bibliografia 26
Rozdział 2. Biometryczna identyfikacja tożsamości na podstawie chodu 28
2.1. Stan wiedzy 29
2.1.1. Metody analizy chodu z wykorzystaniem sieci CNN 29
2.1.2. Baza danych chodu Uniwersytetu McGill 30
2.1.3. Porównanie rezultatów stosowania innych systemów biometrycznych 31
2.1.4. Odtworzenie prędkości kątowej 32
2.1.5. Metody łączenia danych 33
2.2. Metodologia 33
2.2.1. Przetwarzanie sygnałów 34
2.2.2. Segmentacja danych 37
2.2.3. Podział danych na zbiór treningowy i testowy 39
2.2.4. Klasyfikacja chodu 40
Bibliografia 44

Rozdział 3. Zagadnienia segmentacji tęczówki na potrzeby systemów biometrycznych ... 46
3.1. Zagadnienia segmentacji tęczówki oka ... 47
3.1.1. Specyfika obrazu tęczówki oka ... 48
3.1.2. Ogólna struktura systemu rozpoznawania tęczówki ... 50
3.1.3. Zagadnienia segmentacji tęczówki ... 52
3.1.4. Podstawowe metody stosowane w segmentacji tęczówki ... 54
3.2. Badanie wariantów rozwiązań bazujących na algorytmie Daugman'a ... 56
3.2.1. Wykorzystane oprogramowanie ... 56
3.2.2. Metoda oceny jakości segmentacji tęczówki ... 57
3.2.3. Przygotowanie danych testowych ... 59
3.3. Wybrane eksperymenty i wnioski ... 60
3.3.1. Wpływ parametru filtru Gaussa i kroku różniczkowania w operatorze całkująco-różniczkującym ... 60
3.3.2. Zastosowanie algorytmu usuwania refleksów i ograniczenia obszaru poszukiwań ... 62
3.3.3. Wykorzystanie algorytmu poprawy kontrastu ... 64
Bibliografia ... 66
Rozdział 4. Tworzenie uogólnionej architektury wersji oprogramowania z użyciem podobieństw struktur programowych ... 68
4.1. Architektura oprogramowania ... 69
4.2. Odtwarzanie architektury programowania na podstawie kodu źródłowego ... 70
4.3. Linie produkcyjne oprogramowania (SPL) ... 71
4.4. Tworzenie uogólnionej architektury wariantów oprogramowania na podstawie wariantów produktów ... 72
4.5. Analiza klonów ... 74
4.5.1. Klony strukturalne ... 75
4.5.2. Współpracujące klony strukturalne ... 75
4.6. Analiza klonów w tworzeniu uogólnionej architektury ... 76
4.6.1. Użycie analizy klonów w istniejących technikach tworzenia uogólnionej architektury ... 77
4.6.2. Współpracujące klony strukturalne w konstrukcji uogólnionych komponentów ... 79
4.6.3. Bezpośrednie tworzenie uogólnionej architektury z uogólnionych komponentów ... 80
Bibliografia ... 83
Spis tabel ... 86
Spis rysunków ... 87

Wstęp

Pierwsze dwie dekady XXI wieku charakteryzowały się bezprecedensowym wzrostem możliwości gromadzenia danych. Zbiory danych tworzone przez wiele organizacji charakteryzują się wykładniczym wzrostem rozmiaru, między innymi dzięki wykorzystaniu sieci czujników i Internetu Rzeczy (ang. *Internet of Things*). Jednakże same zbiory danych stają się bezużyteczne bez metod analizy i wydobywania z nich informacji. W monografii przedstawiono metody eksploracji zbiorów danych oraz spokrewnione z nimi metody biometryczne.

W rozdziale pierwszym Tomasza Kuczyńskiego *Algorytm k-średnich na wielordzeniowych procesorach CPU oraz akceleratorach GPU* zaprezentowano algorytm grupowania k-średnich, powszechnie uważany za jeden z dziesięciu najważniejszych algorytmów w eksploracji danych. Autor skupił się na porównaniu szybkości pracy algorytmu na współczesnych architekturach równoległych: procesorach wielordzeniowych oraz akceleratorach GPU (ang. *Graphics Processing Unit*). W swoich badaniach wykorzystał nie tylko algorytm naiwny Lloyda, ale również, ostatnio zaproponowane, algorytmy wykorzystujące nierówność trójkąta do znacznego skrócenia czasu obliczeń. Wyniki eksperymentów pokazały, że nowoczesne algorytmy oparte na nierówności trójkąta zaimplementowane na procesorach wielordzeniowych są szybsze niż algorytm Lloyda na akceleratorach GPU.

W rozdziale drugim Aleksandra Sawickiego *Biometryczna identyfikacja tożsamości na podstawie chodu* pokazano zastosowania sieci neuronowych głębokiego uczenia do identyfikacji osób na podstawie chodu. Zaproponowane w pracy, zastosowanie operacji w dziedzinie kwaternionów do wyznaczenia prędkości kątowych na podstawie sygnałów orientacji, pozwoliło na znaczną poprawę wyników klasyfikacji. Uzyskany w ten sposób system biometryczny osiągnął wyniki porównywalne z najlepszymi rezultatami opisanymi w literaturze.

W rozdziale trzecim Mirosława Omieljanowicza *Zagadnienia segmentacji tęczówki na potrzeby systemów biometrycznych* poruszono problem wykorzystania tęczówki oka ludzkiego w biometrii. Autor opisał kolejne przekształcenia, którym podlega obraz tęczówki w systemie biometrycznym, skupiając się głównie na problemie segmentacji. Przedstawił wyniki eksperymentów z klasyczną metodą segmentacji za pomocą operatora całkująco-różniczkującego Daugmana. Zbadał również eksperymentalnie wpływ wielu parametrów metody na skuteczność różnych

wariantów algorytmu segmentacji. Wyniki eksperymentów pokazały, że dodanie do metody dwóch kroków: usuwania refleksów świetlnych oraz poprawy kontrastu metodą adaptacyjnego wyrównywania prowadzi do wzrostu jakości segmentacji tęczówki.

W rozdziale czwartym Cezarego Bołdaka *Tworzenie uogólnionej architektury wersji oprogramowania z użyciem podobieństw struktur programowych* badacz podjął problem zarządzania architekturami oprogramowania posiadającego wiele wersji. Wersje oprogramowania powstają często przez kopiowanie tych samych fragmentów kodu i ich późniejsze modyfikowanie. Prowadzi to do duplikacji fragmentów kodu i tworzenia tzw. klonów programowych oraz związanych z nimi trudności z zarządzaniem architekturą takiego systemu. Autor zaproponował metodę uwzględnienia analizy klonów przy odtwarzaniu uogólnionej architektury wersji oprogramowania.

Wojciech Kwedlo
Białystok, maj 2021

DOI: 10.24427/978-83-66391-84-0_1

Rozdział 1
Algorytm k-średnich na wielordzeniowych procesorach CPU oraz akceleratorach GPU

Tomasz Kuczyński

Streszczenie: Celem procesu grupowania jest podział zbioru danych na klasy o podobnych cechach. Klasy te nazywane są klastrami (skupieniami). Grupowanie dużych zbiorów danych jest procesem czasochłonnym. Jednym z najczęściej wykorzystywanych algorytmów w procesie grupowania danych jest algorytm k-średnich początkowo zaproponowany przez Stuarta Lloyda. Algorytm Lloyda nie jest algorytmem najszybszym. Istnieją algorytmy dające taki sam wynik, po takiej samej liczbie iteracji, ale w dużo krótszym czasie. Główną zaletą algorytmu k-średnich jest dość prosta implementacja. W celu przyspieszenia procesu grupowania danych warto jest wykorzystywać akceleratory GPU, gdyż ich moc obliczeniowa jest większa niż moc obliczeniowa procesorów CPU. W przypadku wykorzystania nowoczesnego akceleratora GPU czas obliczeń może być znacznie krótszy w porównaniu do obliczeń z wykorzystaniem współczesnego procesora CPU (nawet kilkanaście-kilkadziesiąt razy szybciej). Jednym z najpopularniejszych rozwiązań jeżeli chodzi o programowanie układów graficznych jest CUDA (ang. *Compute Unified Device Architecture*). Architektura CUDA opracowana przez firmę NVIDIA i zaprezentowana po raz pierwszy w 2006 roku umożliwia wykorzystanie akceleratorów GPU do obliczeń ogólnego przeznaczenia. W rozdziale zaprezentowane zostaną wyniki przeprowadzonych eksperymentów. Porównane zostaną ze sobą zarówno wyniki obliczeń z wykorzystaniem procesora CPU (algorytm Lloyda oraz algorytmy przyspieszone nierównością trójkąta) jak i wyniki uzyskane przy użyciu akceleratora GPU (algorytm Lloyda).

Słowa kluczowe: algorytm k-średnich, grupowanie, CUDA, nierówności trójkąta

Wprowadzenie

Grupowanie [1] jest jedną z technik uczenia bez nadzoru. Problem grupowania polega na podziale danych na grupy (klastry), które zawierają elementy podobne do siebie. Najpierw dokonywany jest wstępny podział danych. Następnie uzyskany podział jest modyfikowany w taki sposób, że część elementów jest przenoszona do innych grup, tak, aby uzyskać minimalną wariancję wewnątrz każdej grupy. Dąży się do tego, aby podobieństwo elementów w obrębie każdej grupy było jak największe, ale przy jednoczesnym zapewnieniu możliwie maksymalnej różnicy pomiędzy

grupami. Metody grupowania danych można podzielić na dwie kategorie: metody hierarchiczne oraz metody niehierarchiczne. Jedną z metod niehierarchicznych jest metoda k-średnich (ang. *k-means*). Zasadniczą różnicą pomiędzy metodami hierarchicznymi i niehierarchicznymi jest to, iż w przypadku metod niehierarchicznych konieczne jest wcześniejsze podanie liczby grup. Grupowanie danych znajduje szerokie zastosowanie w wielu różnych obszarach (dziedzinach) takich, jak biologia, medycyna, ekonomia, przetwarzanie obrazów.

Za jednorodność elementów w każdej grupie odpowiedzialna jest odpowiednia funkcja oceny. Istnieje wiele różnych funkcji oceny, które mogą być miarą jakości grupowania. Najczęściej stosowana jest funkcja nazywana sumą błędów kwadratowych SSE (ang. *sum of squared error*), którą można zdefiniować jako sumę kwadratów odległości pomiędzy elementami w zbiorze danych a centroidami, do których te elementy są przypisane. Najczęściej stosowaną funkcją odległości jest odległość euklidesowa.

Algorytm k-średnich [2, 3] jest bardzo popularny i powszechnie używany. Dla danego zbioru danych oraz rozwiązań początkowych (centroidów) algorytm w kolejnych iteracjach generuje coraz lepsze rozwiązania, w których wartość SSE jest coraz mniejsza. Iterację algorytmu można podzielić na dwie główne fazy: fazę przypisania oraz fazę obliczeń centroidów. W fazie przypisania każdy element zbioru danych zostaje przypisany do najbliższego centroida. Następnie, w fazie obliczeń centroidów, są one przeliczane jako wartość średnia z przypisanych wektorów uczących. Poszczególne fazy algorytmu są powtarzane, zanim nie zostanie osiągnięte odpowiednie kryterium stopu. Najczęstszym kryterium zbieżności jest z góry przyjęta liczba iteracji lub krok, w którym nie zmieniła się już przynależność elementów zbioru danych do centroidów.

Głównym elementem niniejszej pracy jest porównanie kilku wybranych algorytmów (punkt 1.3). Wykorzystując wielordzeniowy procesor CPU [4, 5] obliczenia przeprowadzono dla następujących algorytmów: Lloyda [2], pierścienia (annulus) [6], Yinyang [7]. Obliczenia z wykorzystaniem akceleratora GPU [8, 9] wykonano dla algorytmu Lloyda. Pozostała część niniejszej pracy wygląda nastepująco. W punkcie 1.1 opisano algorytm k-średnich (algorytm Lloyd'a) oraz omówiono techniki przyspieszania algorytmów k-średnich. Opis platformy testowej wykorzystywanej podczas przeprowadzania eksperymentów zawarto w punkcie 1.2. Pracę zakończono podsumowaniem.

1.1. Algorytm k-średnich

Algorytm k-średnich jest klasycznym algorytmem grupowania danych wynalezionym przez Stuarta Lloyda w 1957 roku i opublikowanym przez niego w 1982 roku [2]. Algorytm Lloyda jest algorytmem siłowym (ang. *brute force*). W każdej iteracji musi zostać obliczona odległość pomiędzy każdym elementem danych (wektorem cech) a każdym centroidem (klastrem). Istnieją algorytmy, które wykorzystują nierówności trójkąta w celu pominięcia obliczeń części odległości. Pozwala to na znaczne skrócenie

czasu obliczeń. Algorytmami, które wykorzystują nierówności trójkąta, są np. algorytm pierścienia (annulus), Yinyang, Elkan, Drake. Należą one do metod, które wprowadzają górną granicę odległości do aktualnie przypisanego centroida oraz dolną granicę odległości do innych centroidów. Nierówności trójkąta umożliwiają aktualizację tych granic, co w rezultacie powoduje, że algorytm potrafi pominąć wiele niepotrzebnych obliczeń odległości, co znacznie wpływa na szybkość działania.

Wartość SSE obliczana jest wg następującego wzoru:

$$SSE = \sum_{i=1}^{N} d^2\left(x(i), c\left(a(i)\right)\right) \tag{1}$$

gdzie:
N – liczba wektorów cech,
d – odległość,
x – wektor cech,
c – centroid,
a – przypisanie wektora cech do klastra.

1.1.1. Algorytm Lloyda

W algorytmie Lloyda [2] (algorytm 1) możemy wyróżnić dwie fazy: fazę przypisania oraz fazę obliczeń centroidów. Fazy te są naprzemiennie wykonywane w poszczególnych iteracjach algorytmu. W fazie przypisania każdy wektor cech zostaje przypisany do najbliższego centroida. Natomiast w kolejnej fazie obliczane są współrzędne centroidów jako suma cech wektorów przypisanych do tego samego klastra (y) podzielona przez liczbę tych wektorów (z). Suma błędów kwadratowych (SSE) zmniejsza się wraz z każdą kolejną iteracją i algorytm dąży do minimum lokalnego. W przypadku, gdy nie zmieniała się już przynależność wektorów cech do centroidów, algorytm kończy swoje działanie. Możliwe jest również zakończenie pracy algorytmu w momencie osiągnięcia określonej liczby iteracji lub z wykorzystaniem kryterium minimalnego względnego ulepszenia SSE.

ALGORYTM 1. Pseudokod algorytmu Lloyda [10]

Dane: Centroidy początkowe $c(1), c(2), ..., c(K)$
i wektory cech $x(1), x(2), ..., x(N)$

```
1  repeat
2      [ faza przypisania ]
3      for i ← 1 to N do
4          a(i) ← arg_j min d(x(i)), c(j))
5      [ faza obliczeń centroidów ]
6      for j ← 1 to K do y(j) ← 0, z(j) ← 0
7      for i ← 1 to N do
```

8 $y(a(i)) \leftarrow y(a(i)) + x(i)$

9 $z(a(i)) \leftarrow z(a(i)) + 1$

10 **for** $j \leftarrow 1$ **to** K **do** $c(j) \leftarrow y(j) / z(j)$

11 **until KryteriumStopu**

Wynik: Centroidy końcowe $c(1)$, $c(2)$, ..., $c(K)$
i przypisania do klastrów $a(1)$, $a(2)$, ..., $a(N)$

Złożoność algorytmu Lloyda:
a) obliczeniowa: $O(iNKD)$ – wartość stała
b) pamięciowa: $O(ND)$
gdzie:
i – liczba iteracji,
N – liczba wektorów cech,
K – liczba centroidów,
D – wymiar

1.1.2. Algorytm pierścienia (annulus)

W algorytmie pierścienia [6], (algorytm 2) górna granica odległości $u(i)$ pomiędzy wektorem cech a przypisanym do niego centroidem musi spełniać następującą nierówność:

$$u(i) \geq \mathrm{d}(x(i), c(a(i))) \tag{2}$$

Natomiast dolna granica odległości $l(i)$ do drugiego najbliższego centroida jest dana nierównością:

$$l(i) \leq \min_{k \neq a(i)} \mathrm{d}(x(k), x(i)) \tag{3}$$

Zmienna (b) jest indeksem drugiego najbliższego centroida przypisanego do danego wektora cech. Jeśli $u(i) \leq l(i)$ to wtedy przypisania do klastrów się nie zmieniają i algorytm pomija najbardziej wewnętrzną pętlę iterującą po centroidach. W celu pominięcia najbardziej wewnętrznej pętli wykonywane są obliczenia wstępne (s) połowy odległości najbliższego centroida do innego najbliższego centroida – wykorzystywany jest tutaj lemat Philipsa [11]. Pętli nie da się pominąć, jeśli granice nakładają się na siebie i sprawdzenie lematu Philipsa kończy się niepowodzeniem. W takim przypadku brane pod uwagę są tylko centroidy, które znajdują się w obszarze pierścieniowym (J), a środkiem tego obszaru jest początek układu współrzędnych.

W fazie przypisania sprawdza się jednocześnie, czy obie granice nie nakładają się na siebie i lemat Philipsa. Jeśli sprawdzenie zakończy się niepowodzeniem, to wtedy zawęża się górną granicę odległości i sprawdza ponownie powyższe warunki. Jeśli sprawdzenie po raz kolejny się nie powiedzie, to przechodzi się do najbardziej wewnętrznej pętli przeszukującej pierścień.

Po zakończeniu fazy przypisania algorytm przechodzi do fazy aktualizacji (obliczeń) centroidów, a następnie przeliczane są granice. Zmienna (δ) oznacza wartość przesunięcia centroida w wyniku wykonania fazy aktualizacji centroidów.

ALGORYTM 2. Pseudokod algorytmu pierścienia (annulus), [10]

Dane: Centroidy początkowe $c(1), c(2), ..., c(K)$
i wektory cech $x(1), x(2), ..., x(N)$

for $i \leftarrow 1$ **to** N **do**
  $a(i) \leftarrow 1, b(i) \leftarrow 1$
  $u(i) \leftarrow \infty, l(i) \leftarrow 0$
repeat
  for $k \leftarrow 1$ **to** K **do** $s(k) \leftarrow \min_{j \neq k} d(c(j)), c(k))/2$
  for $i \leftarrow 1$ **to** N **do**
    $z \leftarrow \max(l(i), s(a(i)))$
    if $u(i) \leq z$ **then continue**
    $u(i) = \mathrm{d}(c(a(i)), x(i))$
    if $u(i) \leq z$ **then continue**
    $l(i) \leftarrow \mathrm{d}(x(i), c(b(i)))$
    $r \leftarrow \max(l(i), u(i))$
    $J \leftarrow \{k:|\mathrm{d}(x(i), 0) - \mathrm{d}(c(k), 0)| < \mathrm{r}\}$
    forall $k \in J$ **do**
      if $\mathrm{d}(x(i), c(k)) < u(i)$ **then**
        [nowy najbliższy centroid]
        $l(i) \leftarrow u(i), u(i) \leftarrow \mathrm{d}(x(i), c(k))$
        $b(i) \leftarrow a(i), a(i) \leftarrow j$
      else if $\mathrm{d}(x(i), c(k)) < l(i)$ **then**
        [nowy drugi najbliższy centroid]
        $b(i) \leftarrow j, l(i) \leftarrow \mathrm{d}(x(i), c(k))$
  $c' \leftarrow c, c \leftarrow$ **FazaObliczeńCentroidów**
  for $k \leftarrow 1$ **to** K **do** $\delta(k) = \mathrm{d}(c(k), c'(k))$
  $\delta' = \max \delta(k)$
for $i \leftarrow 1$ **to** N **do**
  $u(i) \leftarrow u(i) + \delta(a(i))$
  $l(i) \leftarrow l(i) + \delta'$
until KryteriumStopu

Wynik: Centroidy końcowe $c(1), c(2), ..., c(K)$
i przypisania do klastrów $a(1), a(2), ..., a(N)$

Złożoność algorytmu pierścienia (annulus):
a) obliczeniowa: $O(iNKD)$ – wartość pesymistyczna
b) pamięciowa: $O(ND + N)$

1.1.3. Algorytm Yinyang

W algorytmie Yinyang [7], (algorytm 3) górna granica odległości $u(i)$ pomiędzy wektorem cech a przypisanym do niego centroidem musi spełniać następującą nierówność:

$$u(i) \geq \mathrm{d}(x(i), c(a(i))) \tag{4}$$

Natomiast dolna granica odległości $l(i, k)$ pomiędzy wektorem cech a najbliższym centroidem z danej grupy, wyłączając z tego centroidy aktualnie już przypisane, jest dana nierównością:

$$l(i, k) \leq \min_{j \in \pi(k) \wedge j \neq a(i)} \mathrm{d}(c(j), x(i)) \tag{5}$$

ALGORYTM 3. Pseudokod algorytmu Yinyang [10]

Dane: Liczba grup B,
centroidy początkowe $c(1), c(2), ..., c(K)$
i wektory cech $x(1), x(2), ..., x(N)$

$\Pi \leftarrow$ **Partycja** $((c(1), ..., c(K)), B)$
for $i \leftarrow 1$ **to** N **do**
  $a(i) \leftarrow \mathrm{arg}_k \min \mathrm{d}(x(i)), c(k))$
  $u(i) \leftarrow \mathrm{d}(x(i), c(a(i)))$
  for $k = 1$ **to** B **do** $l(i, k) \leftarrow \min_{\{j: j \in \pi(k) \setminus a(i)\}} \mathrm{d}(x(i), c(j))$
repeat
  $c' \leftarrow c$, $c \leftarrow$ **FazaObliczeńCentroidów**
  for $j \leftarrow 1$ **to** K **do** $\delta(j) \leftarrow \mathrm{d}(c(j), c'(j))$
  for $k \leftarrow 1$ **to** B **do** $\Delta(k) \leftarrow \max_{j \in \pi(k)} \delta(j)$
  for $i \leftarrow 1$ **to** N **do**
    $u(i) \leftarrow u(i) + \delta(a(i))$
    $a' \leftarrow a(i)$
    for $k \leftarrow 1$ **to** B **do**

14 $l'(k) \leftarrow l(i, k)$

15 $l(i, k) \leftarrow l(i, k) - \Delta(k)$

16 $L \leftarrow \min_k l(i, k)$

17 **if** $u(i) \leq L$ **then continue**

18 $u(i) \leftarrow \mathrm{d}(x(i), c(a(i)))$

19 **if** $u(i) \leq L$ **then continue**

20 $\Pi \leftarrow \{k : l(i, k) < u(i)\}$

21 **FiltrowanieLokalne**

22 **until KryteriumStopu**

Wynik: Centroidy końcowe $c(1), c(2), ..., c(K)$
i przypisania do klastrów $a(1), a(2), ..., a(N)$

Na samym początku działania algorytmu centroidy zostają podzielone na grupy. Twórcy tego algorytmu zalecają, aby liczba takich grup (B) była nie większa niż $K/10$. Podział rozwiązań początkowych (centroidów) na założoną z góry liczbę grup uzyskuje się przez uruchomienie algorytmu k-średnich tylko na tych początkowych grupach przez pięć iteracji. Pozwala to stworzyć w miarę rozsądne grupy przy poniesieniu jednocześnie niewielkiego narzutu pracy algorytmu.

Następnie, w pierwszej iteracji algorytm dla każdego wektora cech przyporządkowuje górną granicę odległości $u(i)$ oraz dolną granicę odległości $l(i,k)$.

W kolejnym kroku algorytm wykonuje petlę główną, w której: jest dokonywana aktualizacja (obliczenia) centroidów, obliczane jest przesunięcie centroidów $\delta(j)$ oraz wykonywane są obliczenia maksymalnego przesunięcia dla każdej grupy $\Delta(k)$. W pętli iterującej po wszystkich wektorach cech jest aktualizowana górna oraz dolna granica odległości, a następnie, jeśli zajdzie taka potrzeba, to wykonywane są kolejne obliczenia oraz przyporządkowania wektorów cech do centroidów.

W przypadku gdy $u(i) \leq l(i, k)$ obliczenia odległości do wszystkich centroidów z danej grupy nie są konieczne, gdyż wtedy żaden z centroidów nie może znajdować się bliżej danego wektora cech niż centroid aktualnie przypisany. Jeśli $u(i) > l(i, k)$ wtedy algorytm wykonuje obliczenia górnej granicy odległości i jeszcze raz sprawdza nierówność $u(i) \leq l(i, k)$. Jeśli ponownie ta nierówność nie zostanie spełniona, wówczas należy wykonać kolejne obliczenia (filtrowanie lokalne). Algorytm wykonuje lokalne filtrowanie tylko w grupach, które nie spełniły nierówności $u(i) \leq l(i, k)$.

Złożoność algorytmu Yinyang:

a) obliczeniowa: $O(iNKD)$ – wartość pesymistyczna
b) pamięciowa: $O(ND + NB)$

gdzie:
B – liczba grup (najczęściej przyjmuje się $B = K/10$)

1.1.4. Rozwiązania początkowe

Wybór rozwiązań (centroidów) początkowych ma zasadniczy wpływ na efekt końcowy działania algorytmu k-średnich: liczbę iteracji oraz jakość rozwiązania końcowego (wartość SSE). Rozwiązania początkowe najczęściej generowane są w sposób losowy. Można również wykorzystać powszechnie znaną metodę inicjalizacji nazywaną K-means++ [12], która jest inteligentną techniką inicjalizacji centroidów.

Poszczególne kroki metody K-means++ wyglądają następująco:

1) Najpierw wybiera się pierwszego centroida jako losowo wybrany wektor cech ze zbioru danych.
2) Oblicza się kwadraty odległości wszystkich wektorów cech w zbiorze danych od najbliższego wcześniej wybranego centroida.
3) Wybiera się następnego centroida spośród wektorów cech w taki sposób, aby prawdopodobieństwo wyboru wektora cech jako centroida było wprost proporcjonalne do jego kwadratu odległości od najbliższego wcześniej wybranego centroida, tzn. wektor cech mający maksymalną odległość od najbliższego centroida najprawdopodobniej zostanie wybrany jako następny centroid.
4) Powtarza się kroki 2–3, aż znajdziemy wymaganą liczbę centroidów.

1.2. Platforma testowa

Teoretyczna moc obliczeniowa współczesnych akceleratorów GPU jest wielokrotnie większa niż moc obliczeniowa współczesnych procesorów CPU. W niektórych przypadkach obliczeń istnieje możliwość ich przyspieszenia, wykorzystując GPU nawet 100-krotnie względem CPU.

Wydajność dzisiejszego sprzętu (dla pojedynczej precyzji obliczeń), [13]:

a) CPU:
   - Intel i7-8700K CofeeLake – 170 Gflops
   - AMD Ryzen2 2700X Pinnacle Ridge – 198 Gflops
   - Intel i9-9900K CofeeLake-R – 236 Gflops
   - Intel i9-7900X SkyLake-X – 262 Gflops
b) GPU:
   - AMD Radeon RX 480 – 5.8 Tflops
   - AMD Radeon RX 5700 XT – 9.8 Tflops
   - NVIDIA Geforce GTX 1080 Ti – 11 Tflops
   - NVIDIA Geforce RTX 2080 Ti – 13.5 Tflops

Wniosek: Moc obliczeniowa GPU jest większa od mocy obliczeniowej CPU nawet kilkudziesięciokrotnie.

Wszystkie eksperymenty opisane w kolejnym punkcie niniejszej pracy zostały przeprowadzone na jednym węźle znajdującym się w klastrze należącym do Politechniki Białostockiej. Węzeł ten jest komputerem zbudowanym m.in. z procesora AMD Ryzen Threadripper 1950X (16 rdzeni, zegar 3,4 GHz) oraz akceleratora GPU GeForce RTX 2080 Ti (architektura Turing).

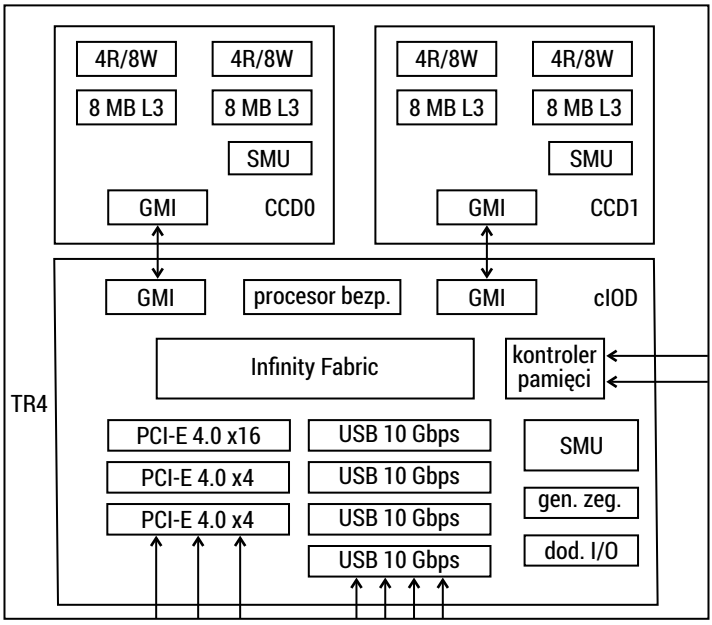
W przypadku obliczeń na procesorze CPU wykorzystano gotową implementację algorytmów autorstwa W. Kwedlo [10, 14]. Trzy algorytmy (Lloyda, pierścienia, Yinyang) zostały zaimplementowane w języku programowania C++ (zrównoleglenie w technologii OpenMP). Kompilacji dokonano przy użyciu kompilatora gcc w wersji 5.4.0.

Wykonując obliczenia z wykorzystaniem akceleratora GPU, bazowano na kodzie zawierającym różne algorytmy grupowania danych (projekt CAMPAIGN, rok 2010) [15]. Wyniki obliczeń oparte o kod projektu CAMPAIGN opublikowano w [16]. Projekt dotyczy generacji akceleratorów pre-Fermi. Algorytmy k-średnich wchodzące w skład projektu zaimplementowano w języku wysokiego poziomu CUDA C. Język ten jest rozszerzeniem języka C/C++ i jednym z najpopularniejszych rozwiązań, jeżeli chodzi o programowanie układów graficznych. Architektura sprzętowo-programistyczna CUDA opracowana przez firmę NVIDIA i zaprezentowana po raz pierwszy w 2006 roku umożliwia wykorzystanie akceleratorów GPU do obliczeń ogólnego przeznaczenia. CUDA jest wieloplatformowym (Windows, Linux, MacOS) i bezpłatnym środowiskiem programowania procesorów firmy Nvidia (kompilator, debuger, profiler, biblioteki) opartym na języku CUDA C. Spośród algorytmów k-średnich dostępnych w projekcie CAMPAIGN do dalszej pracy wybrano jeden algorytm (Lloyda). W kodzie wprowadzono szereg różnorodnych poprawek (np. redukcja warp-synchroniczna wymaga na architekturach >= Volta użycia funkcji syncwarp; usunięto ewidentne błędy typu: nie wszystkie wątki wołają syncthreads) mających na celu dostosowanie go do własnych potrzeb oraz architektur współczesnych akceleratorów GPU. Po wprowadzeniu niezbędnych poprawek dokonano kompilacji przy użyciu kompilatora nvcc w wersji 10.1 (CUDA v10.1).

1.2.1. Architektura procesorów wielordzeniowych na przykładzie procesora AMD Ryzen Threadripper 1950X

Procesor AMD Ryzen Threadripper 1950X jest zbudowany w architekturze Zen [17], (rys. 1.1). Funkcje procesora zostały podzielone pomiędzy dwa lub więcej jąder, które są wykonane w różnych technologiach produkcji. Najwięcej tranzystorów potrzebują rdzenie procesora oraz pamięć podręczna – umieszczono je w jądrach CCD (ang. *Core Chiplet Die*). Natomiast do osobnego jądra produkowanego w innym procesie technologicznym wydzielono części procesora powiązane z interfejsami wejścia/wyjścia, które zawierają dość dużo elementów pasywnych, znoszących wysokie napięcia, których powierzchni nie da się zmniejszyć przy wykorzystaniu najnowszych technik. W jednej obudowie procesora znajdują się dwa lub trzy jądra krzemowe. Procesor umieszczony jest w podstawce TR4. Wewnątrz CCD znajdują się dwa bloki CCX, a w każdym

z nich 4 rdzenie Zen oraz wspólna pamięć podręczna L3. Oprócz rdzeni i pamięci podręcznej znajduje się tam też mikrokontroler SMU (ang. *System Management Unit*) zarządzający zasilaniem, taktowaniem i danymi pochodzącymi z sensorów. Przesył danych do drugiego jądra umożliwia szerokie łącze GMI.



RYSUNEK 1.1. Architektura procesora AMD Ryzen Threadripper 1950X [17]

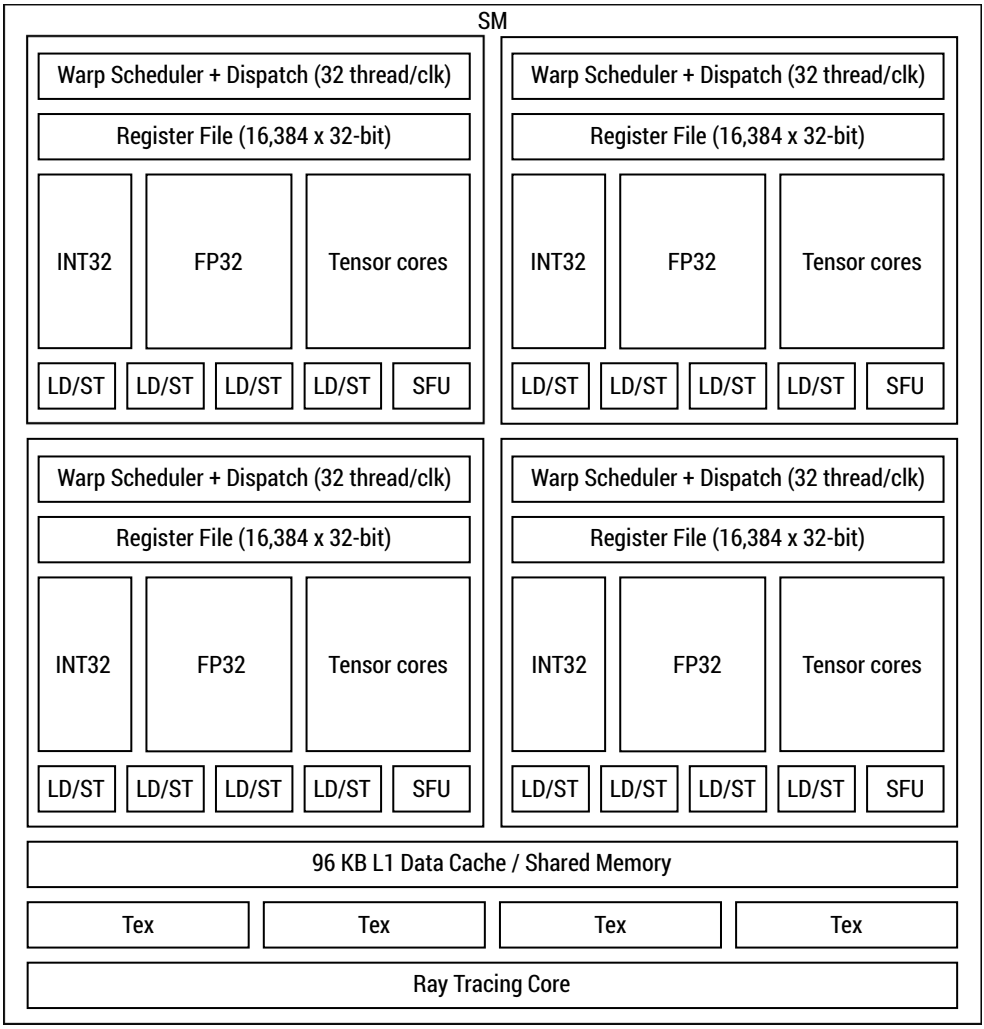
W jądrze cIOD (ang. *coherent I/O Die*) znajduje się kontroler pamięci i kontrolery interfejsów zewnętrznych: 2 x GMI (elektryczna implementacja protokołu Infinity Fabric) do komunikacji z jednym lub dwoma CCD, procesor bezpieczeństwa, 2-kanałowy kontroler pamięci DDR4, kontrolery PCI-E 4.0, kontroler USB 3.2 Gen2x1, SMU (kontroler zasilania i taktowania), generator sygnałów zegarowych, kontrolery innych interfejsów wejścia/wyjścia np. SMBus, eMMC.

1.2.2. Architektura akceleratorów GPU na przykładzie architektury Turing

Firma NVIDIA cały czas opracowuje nowe architektury GPU [18]. Przegląd architektur GPU opracowanych przez firmę NVIDIA oraz daty ich prezentacji przedstawiono poniżej:

- Tesla – listopad 2006
- Fermi – kwiecień 2010
- Kepler – kwiecień 2012
- Maxwell – luty 2014
- Pascal – kwiecień 2016

- Volta – grudzień 2017
- **Turing – wrzesień 2018**
- Ampere – maj 2020.



RYSUNEK 1.2. Schemat budowy bloku SM w architekturze Turing [18]

GPU posiada wiele multiprocesorów, a z kolei każdy multiprocesor ma 64 lub 128 rdzeni CUDA (liczba tych rdzeni jest zależna od architektury GPU np. Turing – 64, Ampere – 128), jednostki przetwarzające specjalnego przeznaczenia i pamięć współdzieloną. Każdy z rdzeni znajdujących się w multiprocesorze jest procesorem wielowątkowym. Poniżej umieszczono opisy poszczególnych jednostek wchodzących w skład bloku SM w architekturze Turing, znajdujących się na rysunku 1.2:

- INT32 – jednostka matematyczna wykonująca operacje na liczbach całkowitych,

- FP32 – jednostka matematyczna przeprowadzająca operacje na liczbach podwójnej precyzji,
- LD/ST – jednostki do odczytu/zapisu danych dla poszczególnych wątków,
- Tex – jednostka tekstur,
- SFU – jednostki przetwarzające specjalnego przeznaczenia, które służą do obliczania funkcji takich jak np. sin, cos, pierwiastek kwadratowy, odwrotność, logarytm.

Multiprocesory wyposażone są w planistów (ang. *Warp Scheduler*).

Architekturę Turing wprowadzono na rynek w 2018 roku i uważana jest za największy skok architektoniczny od ponad dziesięciu lat. W konstrukcji bloków SM (ang. *Streaming Multiprocessor*) wprowadzono dużo zmian. W pojedynczym bloku SM możemy zauważyć, iż poza standardowymi blokami, które wykonują operacje INT32 oraz FP32, dodano dwa dodatkowe bloki wykonawcze (rys. 1.2).

W jednym z tych dodatkowych bloków znajdują się rdzenie Tensor, które odpowiadają za funkcję głębokiego uczenia się (ang. *deep learning*). W drugim dodatkowym bloku umieszczono rdzenie Ray Tracing, które sprzętowo przyspieszają technikę śledzenia promieni (ang. *raytracing*). Wszelkie operacje na liczbach całkowitych oraz zmiennoprzecinkowych dotychczas były wykonywane naprzemiennie. W architekturze Turing te operacje przeprowadzane są równolegle w blokach SM, co powoduje wzrost wydajności. Zmieniono również organizację pamięci podręcznej pierwszego poziomu (L1), pamięci współdzielonej oraz pamięci drugiego poziomu (L2). W architekturze Turing połączono w jeden blok pamięć podręczną pierwszego poziomu (L1) oraz pamięć współdzieloną, co przełożyło się na osiągnięcie dwukrotnie większej przepustowości pomiędzy rdzeniami w bloku SM a pamięcią podręczną. W wyniku takiego połączenia pamięć podręczna pierwszego poziomu oraz współdzielona mogą wysyłać dużo większe porcje danych do pamięci podręcznej drugiego poziomu tzn. odpowiednio o rozmiarze 32 kB i 64 kB. Poza tym, zwiększono dwukrotnie rozmiar pamięci podręcznej drugiego poziomu (do 6 MB). Pamięć GDDR5X zastąpiono nowym systemem pamięci GDDR6, który jest wspierany przez 16 kontrolerów. Prędkość przesyłania danych wzrosła z 11 Gb/s do 14 Gb/s. W architekturze Turing zastosowano również po raz pierwszy (w architekturze zorientowanej na użytkownika) niezależne programowanie wątków, tzn. wątki mogą być programowane w warp przez SM bez potrzeby oczekiwania na ich możliwie najszybszą zbieżność.

1.3. Eksperymentalne porównanie algorytmów

W niniejszym punkcie zostaną przedstawione wyniki przeprowadzonych eksperymentów dla czterech wybranych algorytmów. Do eksperymentów wybrano trzy zbiory danych 10-wymiarowe o rosnącej liczbie wektorów (mix10_1.5mln, mix10_15mln, mix10_150mln) oraz trzy zbiory 300-wymiarowe (mix300_50tys, mix300_500tys, mix300_5mln). Pierwszy człon w nazwie zbioru to wymiar, a drugi człon to liczba

wektorów. Liczbę wektorów dobrano tak, aby całkowita liczba elementów macierzy danych była taka sama dla zbioru najmniejszego, średniego i największego. Dane w tych zbiorach tworzą dobrze odseparowane klastry z wielowymiarowego rozkładu normalnego (Gaussa) – jest to tak zwana mieszanina gaussowska (ang. *Gaussian Mixture Model*). Obrano następujące liczby centroidów: K = { 4, 16, 64, 256, 1024, 4096}. Reprezentacja centroidów jest pojedynczej precyzji (float). Dla każdego zbioru danych w powiązaniu z konkretną liczbą centroidów wygenerowano jedno rozwiązanie początkowe przy użyciu algorytmu k-means|| [19].

Dla każdej kombinacji zbioru danych oraz liczby centroidów uruchomiono trzy algorytmy k-średnich na procesorze CPU (Lloyda, pierścienia, Yinyang) oraz jeden algorytm na akceleratorze GPU (Lloyda). Ze względu na błędy zaokrągleń wynikiem jest średni czas iteracji. Zmierzone czasy mogą jednak różnić się nawet dość znacznie, gdy uruchomimy ten sam algorytm kilkukrotnie na tym samym sprzęcie. Związane jest to z występowaniem szumu systemowego. W związku z tym każdy eksperyment powtórzono trzy razy. Ostatecznym wynikiem jest mediana z trzech średnich czasów iteracji. Wszystkie wyniki zestawiono w tabeli 1.1 oraz 1.2. Jako kryterium stopu dla każdego z algorytmów przyjęto moment, w którym względna poprawa wartości SSE między dwiema kolejnymi iteracjami była mniejsza niż 10^{-6}. Oprócz średnich czasów iteracji wyznaczono również przyspieszenie algorytmiczne danego algorytmu A względem algorytmu Lloyda zgodnie z wzorem (6):

$$S_A = \frac{t_{Lloyd}}{t_A} \tag{6}$$

Pozioma linia znajdująca się przy 10^0 na rysunkach oznacza przyspieszenie algorytmiczne nad algorytmem Lloyda (CPU) wynoszące 1.

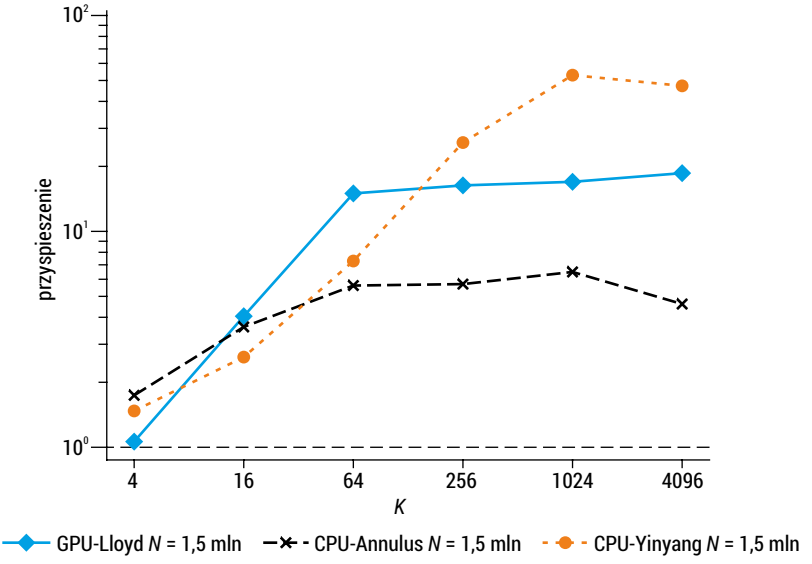
TABELA 1.1. Średnie czasy iteracji (w sekundach) dla różnych algorytmów k-średnich w zależności od liczby klastrów K oraz liczby wektorów uczących N (zbiory mix10)

K	N	Lloyd CPU	Lloyd GPU	Annulus CPU	Yinyang CPU
4	1,5 mln	0,01519	0,01431	0,008734	0,01031
	15 mln	0,1201	0,1241	0,06525	0,06631
	150 mln	1,140	1,237	0,6994	0,7231
16	1,5 mln	0,04528	0,01119	0,01254	0,01730
	15 mln	0,4149	0,1029	0,1369	0,1733
	150 mln	4,096	1,229	1,366	1,738
64	1,5 mln	0,1604	0,01070	0,02859	0,022
	15 mln	1,564	0,1012	0,2615	0,2167
	150 mln	15,57	1,032	2,712	2,297
256	1,5 mln	0,6148	0,03765	0,1078	0,02383
	15 mln	6,118	0,3765	0,9502	0,2092
	150 mln	61,17	3,755	8,188	1,989

K	N	Lloyd CPU	Lloyd GPU	Annulus CPU	Yinyang CPU
1024	1,5 mln	2,434	0,1434	0,3752	0,04599
	15 mln	24,30	1,441	3,186	0,4157
	150 mln	243	14,52	25	3,328
4096	1,5 mln	9,701	0,5212	2,106	0,2054
	15 mln	96,99	5,185	12,11	1,34
	150 mln	969,8	52,07	114,3	9,968

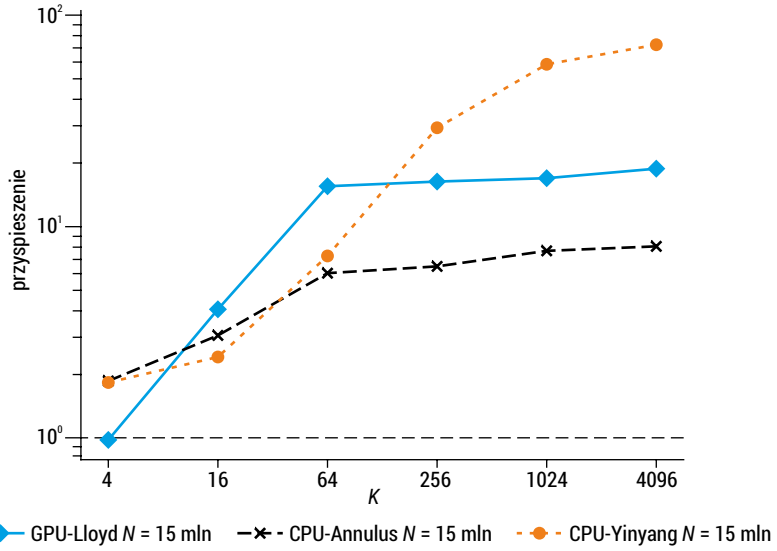
Na rysunkach 1.3, 1.4 i 1.5 przedstawiono wyniki obliczeń z wykorzystaniem zbiorów danych 10-wymiarowych. Każdy z tych trzech rysunków dotyczy innej wartości N, czyli liczby wektorów cech (1.5 mln, 15 mln, 150 mln). Przedstawiono przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyd'a w wersji na CPU dla różnych wartości K.

Na rysunku 1.3, czyli w przypadku najmniejszego spośród zbiorów 10-wymiarowych, widać, że dla K = 4 najszybszy jest algorytm pierścienia, dla K = 16 i 64 najszybszy jest algorytm Lloyd'a na GPU, a dla największych wartości K = 256, 1024 oraz 4096 algorytm Yinyang jest bezkonkurencyjny. Generalnie przewaga wszystkich trzech algorytmów nad algorytmem Lloyda w wersji na CPU rośnie wraz ze wzrostem liczby klastrów K. Jedynie w przypadku największej liczby klastrów przewaga ta już nie jest rosnąca dla algorytmów pierścienia oraz Yinyang. Największe przyspieszenie osiąga algorytm Yinyang dla wartości K = 1024 oraz 4096 i wynosi ono odpowiednio 52,93 i 47,23.



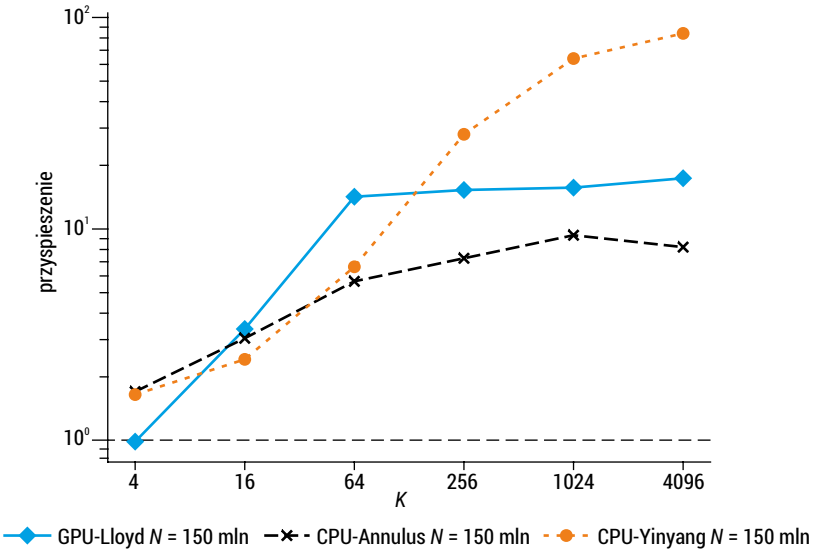
RYSUNEK 1.3. Przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 1,5 mln

Rysunek 1.4 przedstawia wyniki obliczeń dla średniego zbioru danych pod względem wielkości spośród trzech zbiorów 10-wymiarowych. Szybkość algorytmów dla poszczególnych wartości K, jest identyczna jak w przypadku najmniejszego zbioru danych, tzn. dla K = 256, 1024 oraz 4096 najszybszy jest algorytm Yinyang, a dla mniejszych wartości K najszybszy jest algorytm Lloyda na GPU lub pierścienia. Tutaj również można zaobserwować wzrost przewagi trzech algorytmów nad algorytmem Lloyda na CPU wraz ze wzrostem liczby klastrów K. Algorytm Yinyang osiąga największe przyspieszenie wynoszące 58,46 oraz 72,39 odpowiednio dla K = 1024 i 4096 – są to wartości większe niż w przypadku zbioru najmniejszego.



RYSUNEK 1.4. Przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 15 mln

Na rysunku 1.5 przedstawiono wyniki obliczeń dla największego zbioru danych spośród zbiorów 10-wymiarowych. Identycznie jak w przypadku dwóch mniejszych zbiorów danych, algorytm Yinyang jest najszybszy dla K = 256, 1024 oraz 4096, a dla mniejszych wartości K najszybszy jest algorytm Lloyda na GPU lub pierścienia. Wraz ze wzrostem liczby klastrów K tutaj również zauważalny jest wzrost przewagi trzech algorytmów nad algorytmem Lloyda na CPU. Największe przyspieszenie wynoszące 73,02 oraz 97,29 osiąga algorytm Yinyang odpowiednio dla K = 1024 i 4096.



RYSUNEK 1.5. Przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 150 mln

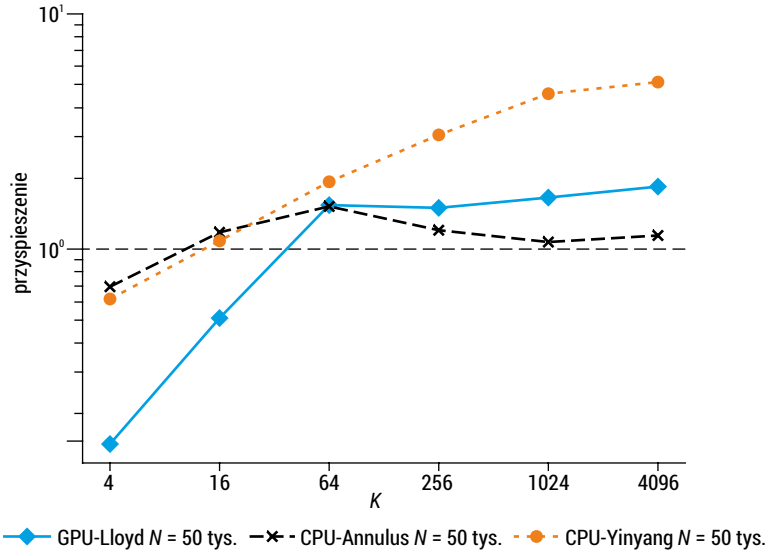
Kolejne trzy rysunki (rys. 1.6, 1.7 i 1.8) przedstawiają wyniki obliczeń dla zbiorów danych 300-wymiarowych. Każdy rysunek jest związany z inną wartością N (50 tys., 500 tys., 5 mln). Dla różnych wartości K przedstawiono przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyda na CPU.

TABELA 1.2. Średnie czasy iteracji (w sekundach) dla różnych algorytmów k-średnich w zależności od liczby klastrów K oraz liczby wektorów uczących N (zbiory mix300)

K	N	Lloyd CPU	Lloyd GPU	Annulus CPU	Yinyang CPU
4	50 tys 500 tys 5 mln	0,001619 0,01836 0,1411	0,01101 0,09544 0,9203	0,002361 0,02303 0,2057	0,002662 0,02567 0,2286
16	50 tys 500 tys 5 mln	0,005197 0,04866 0,4399	0,01029 0,09497 0,8708	0,004446 0,03811 0,3931	0,004834 0,04192 0,4358
64	50 tys 500 tys 5 mln	0,02141 0,1699 1,644	0,01403 0,1289 1,352	0,01421 0,09906 0,9351	0,01117 0,08932 0,7790
256	50 tys 500 tys 5 mln	0,07159 0,6509 6,467	0,04821 0,4690 4,911	0,06001 0,4835 4,831	0,02360 0,1794 1,742

K	N	Lloyd CPU	Lloyd GPU	Annulus CPU	Yinyang CPU
1024	50 tys 500 tys 5 mln	0,2670 2,59 25,95	0,1625 1,63 17,98	0,2513 2,496 21,53	0,05881 0,61 4,479
4096	50 tys 500 tys 5 mln	1,116 10,38 103,8	0,6102 6,047 69,38	0,9851 10,24 103,1	0,2193 2,27 18,43

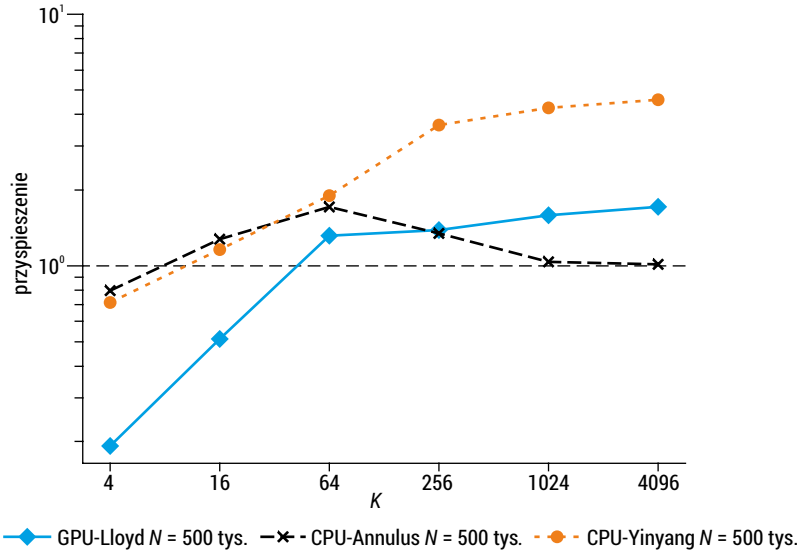
Wyniki obliczeń dla najmniejszego spośród zbiorów 300-wymiarowych przedstawiono na rysunku 1.6. Dla K = 4 oraz 16 najszybszy jest algorytm pierścienia (przewaga nad algorytmem Yinyang jest niewielka), a dla pozostałych wyższych wartości K (64, 256, 1024 oraz 4096) algorytm Yinyang jest najszybszy. Przewaga algorytmów Yinyang oraz Lloyda (GPU) nad algorytmem Lloyda w wersji na CPU zwiększa się wraz ze wzrostem liczby klastrów K. Natomiast w przypadku algorytmu pierścienia przyspieszenie osiąga wartość maksymalną dla K = 64, po czym wartość przyspieszenia zaczyna zmniejszać się wraz ze wzrostem K. Największe przyspieszenie osiąga algorytm Yinyang dla wartości K = 1024 oraz 4096 i wynosi ono odpowiednio 4,54 i 5,09.



RYSUNEK 1.6. Przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 50 tys

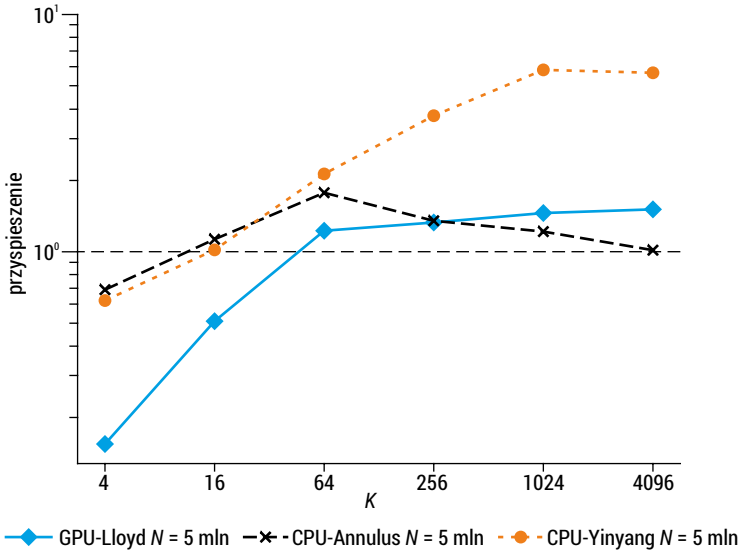
Na rysunku 1.7 przedstawiono wyniki obliczeń dla średniego zbioru danych spośród zbiorów 300-wymiarowych. Identycznie jak w przypadku najmniejszego zbioru danych, algorytm pierścienia jest najszybszy dla K = 4 i 16, a dla większej liczby klastrów najszybszym algorytmem jest algorytm Yinyang. Wraz ze wzrostem liczby klastrów K zwiększa się również przewaga algorytmów Yinyang oraz Lloyda

(GPU) nad algorytmem Lloyda (CPU). Podobnie jak w przypadku najmniejszego zbioru danych, algorytm pierścienia największe przyspieszenie osiąga dla K = 64, a potem dla większych wartości K przyspieszenie już tylko zmniejsza się i dla K = 1024 oraz 4096 przewaga tego algorytmu nad algorytmem Lloyda (CPU) jest znikoma. Największe przyspieszenie wynoszące 4,25 oraz 4,57 osiąga algorytm Yinyang odpowiednio dla K = 1024 i 4096.



RYSUNEK 1.7. Przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 500 tys

Rysunek 1.8 przedstawia wyniki obliczeń dla największego zbioru danych pod względem wielkości spośród trzech zbiorów 300-wymiarowych. Szybkości algorytmów dla poszczególnych wartości K, jest identyczna jak w przypadku dwóch mniejszych zbiorów danych, tzn. dla K = 4 oraz 16 najszybszy jest algorytm pierścienia, a dla większych wartości K najszybszy jest algorytm Yinyang. Przewaga algorytmów Yinyang oraz Lloyda (GPU) nad algorytmem Lloyda (CPU) zwiększa się wraz ze wzrostem liczby klastrów K. Identycznie jak w przypadku dwóch mniejszych zbiorów danych, algorytm pierścienia największe przyspieszenie osiąga dla K = 64, a dla większych wartości K przyspieszenie zmniejsza się i dla K = 4096 przewaga tego algorytmu nad algorytmem Lloyda (CPU) jest już znikoma. Algorytm Yinyang osiąga największe przyspieszenie wynoszące 5,79 oraz 5,63 odpowiednio dla K = 1024 i 4096 – są to wartości większe niż w przypadku dwóch mniejszych zbiorów.



RYSUNEK 1.8. Przyspieszenie trzech algorytmów k-średnich nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 5 mln

Podsumowanie

Podstawowym celem niniejszej pracy było porównanie czterech algorytmów. Na mocy przeprowadzonych eksperymentów można stwierdzić, że najszybszym algorytmem spośród testowanych okazał się algorytm Yinyang, którego przewaga nad innymi jest szczególnie widoczna dla K ≥ 256. W przypadku mniejszych wartości K czasami szybszy okazywał się algorytm Lloyda (GPU) lub pierścienia, ale ta przewaga nad algorytmem Yinyang była niewielka. Bardzo duży wpływ na przyspieszenie ma też wymiar przestrzeni cech oraz centroidów. Przy wektorach cech i centroidach 10-wymiarowych przyspieszenie może wynieść nawet około 100 (algorytm Yinyang, N = 150 mln, K = 4096). Natomiast w przypadku wektorów cech i centroidów 300-wymiarowych maksymalne przyspieszenie wynosi zaledwie około 6 (algorytm Yinyang, N = 5 mln, K = 1024). Wyraźnie widać, że testowane algorytmy niezbyt dobrze radzą sobie z dużymi wymiarami. Wielkość zbioru danych (liczba wektorów cech N) również może mieć zauważalny wpływ na przyspieszenie przy większych wartościach K, np. 1024 lub 4096. W szczególności jest to widoczne w przypadku zbiorów 10-wymiarowych. Wtedy wraz ze wzrostem liczby wektorów cech w zbiorze danych wzrasta również zauważalnie wartość przyspieszenia. W przypadku zbiorów danych 300-wymiarowych jest to bardzo mało widoczne. Algorytm pierścienia wyjątkowo słabo radzi sobie ze zbiorami danych 300-wymiarowymi dla K = 1024 i 4096 – wtedy jego przewaga nad algorytmem Lloyda (CPU) jest bardzo mała bądź nawet znikoma.

W celu poprawy rezultatów warto rozważyć dalsze prace nad algorytmem Lloyda na GPU, gdyż w kodzie bazującym na projekcie CAMPAIGN zbyt wolno działa faza przypisania i zajmuje ona > 95% czasu pracy algorytmu. Innym podejściem mogłoby być opracowanie w wersji na akcelerator GPU któregoś z algorytmów wykorzystujących nierówności trójkąta. Z dwóch przetestowanych algorytmów opierających się na tych zależnościach lepszym wyborem byłby algorytm Yinyang.

Badania zostały zrealizowane w ramach pracy nr WZ/WI-IIT/3/2020, sfinansowanej ze środków Ministerstwa Nauki i Szkolnictwa Wyższego w Polsce.

Bibliografia

[1] Jain A. K., *Data clustering: 50 years beyond K-means*, Pattern Recognition Letters, 2010, t. 31, nr 8, pp. 651–666

[2] Lloyd S., *Least squares quantization in PCM*, IEEE Transactions on Information Theory, 1982, t. IT 28, nr 2, pp. 129–137

[3] MacQueen J., *Some methods for classification and analysis of multivariate observations*, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967

[4] Dhillon I.S. i Modha D.S., *A data-clustering algorithm on distributed memory multiprocessors*, Large-Scale Parallel Data Mining, Berlin, Germany, Springer, 2002, pp. 245–260

[5] Rodrigues L.M., Zárate L.E., Nobre C.N. i Freitas H.C., *Parallel and distributed kmeans to identify the translation initiation site of proteins*, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2012

[6] Hamerly G. i Drake J., *Accelerating Lloyd's algorithm for k-means clustering*, Partitional Clustering Algorithms, Cham, Switzerland, Springer, 2015, pp. 41–78

[7] Ding Y., Zhao Y., Shen X., Musuvathi M. i Mytkowicz T., *Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup*, Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015

[8] Li Y., Zhao K., Chu X. i Liu J., *Speeding up k-means algorithm by GPUs*, Journal of Computer and System Sciences, 2013, t. 79, nr 2, pp. 216–229

[9] Cuomo S., De Angelis V., Farina G., Marcellino L. i Toraldo G., *A GPU-accelerated parallel K-means algorithm*, Computers and Electrical Engineering, 2019, t. 75, pp. 262–274

[10] Kwedlo W. i Czochanski P., *A Hybrid MPI/OpenMP Parallelization of K-Means Algorithms Accelerated Using the Triangle Inequality*, IEEE Access, 2019, t. 7, pp. 42280–42297

[11] Phillips S., *Acceleration of k-means and related clustering algorithms*, Algorithm Engineering and Experiments: 4th International Workshop, Springer, 2002, t. 2409, pp. 166–177

[12] Arthur D. i Vassilvitskii S., *K-means++: The advantages of careful seeding*, Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007

[13] TechPowerUp, *TechPowerUp*, 1 Grudzień 2020. [Online]. Available: www.techpowerup.com

[14] Kwedlo W., *Two modifications of Yinyang K-means algorithm*, Lecture Notes in Computer Science, t. 10246, Cham, Switzerland, Springer, 2017, pp. 94–103

[15] SimTK, *Clustering Algorithms for Massively Parallel Architectures Including GPU Nodes*, 1 Grudzień 2020. [Online]. Available: simtk.org/projects/campaign

[16] Kohlhoff K.J., Pande V.S. i Altman R.B., *K-Means for Parallel Architectures Using All-Prefix-Sum Sorting and Updating Steps*, IEEE Transactions on Parallel and Distributed Systems, 2013, t. 24, nr 8, pp. 1602–1612

[17] PCLab, *PCLab*, 19 Styczeń 2021. [Online]. Available: pclab.pl

[18] ITIGIC, *ITIGIC*, 19 Styczeń 2021. [Online]. Available: itigic.com/pl/

[19] Bahmani B., Moseley B., Vattani A., Kumar R. i Vassilvitskii S., *Scalable k-means++*, Proceedings of the VLDB Endowment, 2012, t. 5, nr 7, pp. 622–633

DOI: 10.24427/978-83-66391-84-0_2

Rozdział 2
Biometryczna identyfikacja tożsamości na podstawie chodu

Aleksander Sawicki

Streszczenie: W artykule zawarto wyniki oryginalnych badań nad zastosowaniem splotowych sieci neuronowych w identyfikacji osób na podstawie chodu. W pracy wykorzystano ogólnodostępny korpus danych, zawierający sygnały sensorów IMU (ang. *Inertial Measurement Unit*) takie, jak przyspieszenie czy natężenia pola magnetycznego.

W przeprowadzonych badaniach nieprzetworzone wartości pomiarowe sensorów (ang. *raw data*) podano bezpośrednio na wejście klasyfikatorów CNN (ang. *Convolution Nerual Network*). Jednocześnie zastosowano technikę oddzielania zerami (ang. *zero-padding*) do przedzielenia danych pochodzących z sensorów różnego typu. Podkreślić należy, iż podejście takie stosowane było wcześniej, przede wszystkim w zagadnieniach rozpoznawania czynności (ang. *Human Activity Recognition*). Dużą wartość dodaną pracy przedstawiają operacje w dziedzinie kwaternionów, umożliwiające wyznaczanie prędkości kątowych na podstawie sygnałów orientacji. Wykorzystanie sygnałów prędkości kątowych pozwoliło na osiągnięcie bardzo dobrych rezultatów klasyfikacji w finalnym systemie biometrycznym.

Zastosowanie techniki oddzielenia zerami (ang. *zero padding*) pozwoliło osiągnąć obiecujące wskaźniki identyfikacji osób w zagadnieniu biometrii behawioralnej. Osiągnięte wyniki były porównywalne z rezultatami opisywanymi w literaturze.

Słowa kluczowe: Biometria, Klasyfikacja, IMU

Wprowadzenie

Zadaniem systemów biometrycznych jest rozróżnianie osób na podstawie wybranych cech fizjologicznych lub behawioralnych. Badania nad rozwojem i wdrażaniem tego typu rozwiązań przyciągają uwagę zarówno środowisk akademickich, jak i przemysłowych z uwagi na rosnące zapotrzebowanie identyfikacji i weryfikacji osób [1]. Systemy biometryczne bazujące na analizie chodu, w przeciwieństwie do tych wykorzystujących dane wejściowe w postaci m.in: zdjęć tęczówki oka, żył dłoni czy odcisku

palca, mogą działać w sposób bardziej dyskretny [2]. W ostatnim czasie zaobserwować można rozwój tej szczególnej dziedziny biometrii behawioralnej, właśnie ze względu na brak konieczności aktywnej kooperacji z uczestnikiem badań [3].

W rozdziale przedstawiono rezultaty pracy nad systemem biometrycznej identyfikacji osób na podstawie chodu. Zaproponowana metodologia bazuje na przetwarzaniu nieprzetworzonych wartości pomiarowych sensorów ruchu IMU (ang. *Inertial Measurement Unit*) za pomocą sztucznych splotowych sieci neuronowych CNN (ang. *Convolution Neuranl Network*).

Prezentowane eksperymenty przeprowadzono z wykorzystaniem ogólnodostępnej bazy chodu ludzkiego, zgromadzonej w uniwersytecie McGill [4]. Korpus danych zawiera zestaw nagrań ruchu dwudziestu uczestników badań. Proces akwizycji danych przeprowadzony został z wykorzystaniem telefonu typu smartphone, wyposażonego w czujniki takie, jak akcelerometr i magnetometr.

Klasyfikacja chodu ludzkiego przeprowadzona została z wykorzystaniem sieci neuronowych CNN. Na wejścia klasyfikatora poddano nieprzetworzone wielkości pomiarowe sensorów. Dane te oddzielone zostały blokami zerowych wartości, aby nie doszło do ich interferencji w pierwszej warstwie splotowej. W tym celu zastosowano technikę uzupełniania zerami (ang. *zero padding*), [5]. W wyniku przeprowadzonych badań zaproponowano architekturę systemu biometrycznego, umożliwiającego identyfikację uczestników badań na poziomie około 90%.

Dużą wartość dodaną pracy stanowi zestaw procedur w dziedzinie algebry kwaternionów, umożliwiających odtworzenie sygnałów prędkości kątowej na podstawie sygnałów orientacji. Rozwiązanie tego typu zwiększyło skuteczność opracowanego systemu biometrycznego.

2.1. Stan wiedzy

Przegląd stanu wiedzy dotyczył kilku pomniejszych zagadnień i został podzielony na następujące punkty: analiza chodu z wykorzystaniem sieci CNN, opis wykorzystanej bazy danych, zestawienie skuteczności systemów biometrycznych innych zespołów naukowych, metody odtwarzania prędkości kątowej, metody analizy danych różnych grup sensorów przez sieci splotowe.

2.1.1. Metody analizy chodu z wykorzystaniem sieci CNN

Zagadnienia związane z biometrią behawioralną w znacznie większej mierze dotyczą analizy ludzkiego chodu niż jakichkolwiek innych ruchów czy też gestów. Współcześnie wyodrębnić można dwa główne nurty dotyczące zagadnień analizy ruchu z wykorzystaniem sztucznych splotowych sieci neuronowych. W pierwszym z nich autorzy wykorzystują wyniki różnorodnych transformat, doprowadzających

do powstania „obrazów" dwuwymiarowych. W publikacji [3] wykorzystano nowatorską technikę nazwaną AE-GD (ang. *Angle Embedded Gait Dynamic Image*). Polegała ona na wyznaczeniu macierzy dwuwymiarowych na podstawie sygnału magnitudy akcelerometru. W przypadku publikacji [6] autorzy zastosowali transformatę czasowo-częstotliwościową w postaci Dystrubucji Wignera-Villa. Odmienne aczkolwiek bardzo podobne podejście zaprezentowali autorzy [7] wykorzystując scalogramy – wyniki transformaty falkowej.

W drugiej grupie rozwiązań zamieścić można rozwiązania bazujące bezpośrednio na wartościach pomiarowych IMU. Reprezentantem tej grupy może być publikacja [8], w której autorzy wykorzystali surowe wartości akcelerometru i żyroskopu oraz sieci splotowe w formie ekstraktora danych. Z kolei w publikacji [9] wykorzystano architekturę sieci neuronowej o wielu wejściach w celu identyfikacji osoby wykonującej wybrane gesty.

2.1.2. Baza danych chodu Uniwersytetu McGill

Opisywane w rozdziale badania dotyczą zagadnienia biometrii behawioralnej ludzkiego chodu. W przeprowadzonych eksperymentach wykorzystano ogólnodostępną bazę [4], zgromadzoną w uniwersytecie McGill. W tworzeniu korpusu danych wzięło udział 20 uczestników, odpowiednio 10 kobiet i 10 mężczyzn. Poproszeni zostali oni o udział w dwóch około 15-minutowych sesjach nagrywania chodu, które odbywały się w różnych dniach. Podkreślić należy, iż na potrzeby opisywanych w rozdziale badań wykorzystano jedynie dane zarejestrowane podczas pierwszej z nich.

Akwizycja danych przeprowadzona została z wykorzystaniem sensorów ruchu wbudowanych w telefon komórkowy typu smartphone (urządzenia HTC Nexus One). W celu odczytu danych pomiarowych czujników ruchu opracowano oraz wykorzystano autorskie oprogramowanie. Z uwagi na akwizycję danych z udziałem sensorów działających pod systemem operacyjnym, akwizycja danych przebiegała z częstotliwością próbkowania wynoszącą 27,52 ± 4,07 Hz.

Wśród udostępnionych w korpusie danych sygnałów znajdują się::

- znacznik czasowy (ang. *timestamp*);
- wartości pomiarowe akcelerometru (odpowiednio osie *OX*,*OY*,*OZ* oraz magnituda);
- wartości pomiarowe magnetometru (odpowiednio osie *OX*,*OY*,*OZ*);
- informacje o orientacji telefonu w postaci kątów Eulera tj; *Yaw*, *Pitch Roll*.

Osoby biorące udział w eksperymencie proszone były o typowe dla nich użytkowanie telefonu komórkowego. W związku z czym w zależności od preferencji uczestnika, sensory umieszczone były w prawej lub lewej oraz przedniej lub tylnej kieszeni spodni. Jednocześnie osoby biorące udział w eksperymencie w dowolnym momencie mogły wyjąć telefon, a następnie włożyć go z powrotem, niekoniecznie zachowując oryginalne ułożenie urządzenia. Nagrania zawierają również fragmenty małej

aktywności (bezruchu), w czasie których czujniki ruchu nie wykrywały znaczących zmian. Domniemywać można, że w tych okresach uczestnicy badań np. rozmawiali z osobami spotkanymi na terenie kampusu.

Finalnie zaznaczyć należy, iż poszczególne osoby posiadały różne typy obuwia, chód ich zarejestrowany został na terenie kampusu akademickiego, przy czym nie zachowywali oni stałej ścieżki spacerów. Ponadto, proces akwizycji danych odbył się z udziałem czterech różnych egzemplarzy telefonów komórkowych.

2.1.3. Porównanie rezultatów stosowania innych systemów biometrycznych

Korpus danych wykorzystywany w pracy był kilkukrotnie używany w publikacjach naukowych z uwagi na swój ogólnodostępny charakter.

W pozycji [3] autorzy opisali opracowany algorytm generowania form pseudo-obrazów *GDS* (ang. *Gait Dynamics Images*) na podstawie sygnałów pomiarowych trzech osi akcelerometrów. Prezentowane metody zakładały całkowite pominięcie innych sygnałów dostępnych w bazie. Prezentowany system biometryczny wykorzystywał grafiki wygenerowane na podstawie okien przesuwnych o długości 2,4 sekund. Do oceny podobieństwa sygnałów testowych oraz wzorcowych wykorzystano sumaryczne szczytowe współczynniki korelacji. W pracy wykorzystano klasyfikator w postaci algorytmu najbliższych sąsiadów. Ocena jakości identyfikacji osób odbyła się na podstawie wyciętych bloków danych o długości 50 sekund przy wykorzystaniu okien przesuwnych o długości 2,4 sekund oraz nakładkowaniu wynoszącym 50%. Jako miarę jakości klasyfikacji wykorzystano skuteczność przy zastosowaniu prostej walidacji.

Z kolei w pracy [3] zaproponowano zmodyfikowany sposób generowania obrazów nazwany jako AE-GDI (ang. *Angle Embedded Gait Dynamic*). Przedstawiona metoda również bazowała na graficznej reprezentacji wygenerowanej na podstawie sygnałów pomiarowych akcelerometru. W przeciwieństwie do poprzedniej pozycji [4] autorzy zaimplementowali algorytm detekcji poszczególnych cykli chodu. Opracowany system biometryczny wykorzystywał grafiki wygenerowane na podstawie sygnałów pojedynczego cyklu chodu lub próbek dwóch sklejonych przebiegów. W pracy wykorzystano klasyfikator w postaci sieci splotowej CNN. Testowanie przebiegło w wysublimowanych warunkach. Z dostępnych w ciągu jednego dnia przebiegów czasowych wyznaczano blok o długości 10% całości, stanowiący zbiór treningowy. Następnie dokonywano zaawansowanej segmentacji cykli chodu oraz generowano grafiki. Zbiór testowy był generowany losowo na podstawie niewykorzystanych danych i posiadał tę samą długość co zbiór treningowy (tj. 10% całości). Proces ten powtarzany był dziesięciokrotnie. Jako miarę jakości klasyfikacji wykorzystano skuteczność. Porównanie jakości klasyfikacji poszczególnych metod zaprezentowano w tabeli 2.1.

TABELA 2.1. Porównanie skuteczności systemów biometrycznych bazujących na bazie [4]

forma danych wejściowych	Metoda	Skuteczność [%]
bloki danych	Bazowa [4] (oparta na magnitudzie akcelerometru)	67,5
bloki danych	GDI [4]	87,5
bloki danych	Zmodyfikowana metoda GDI [4]	85,5
cykle chodu	AE-GDI (pojedynczy cykl chodu) [3]	95,4
cykle chodu	AE-GDI (podwójny cykl chodu) [3]	96,6

2.1.4. Odtworzenie prędkości kątowej

Współcześnie, z uwagi na wzrost aplikacji metod sztucznej inteligencji z dziedziny uczenia głębokiego, szczególnie wymagających licznych zbiorów uczących, zaobserwować można zwiększenie liczebności prac naukowych w dziedzinie augmentacji danych, jak i generowania próbek syntetycznych. Nurt ten widoczny jest również w dziedzinie przetwarzania sygnałów ruchu.

W pracach [10,11] badacze zaproponowali autorskie rozwiązania umożliwiające odtworzenie sygnałów akcelerometru oraz żyroskopu. W obu tych przypadkach zastosowano modele regresyjne umożliwiające zasymulowanie wskazań sensorów na podstawie animacji szkieletowych, wyodrębnionych z nagrań wideo lub też nagrań kamery głębi. W obu opisywanych algorytmach niezbędne było wykorzystani obu modalności w celu wcześniejszej nauki modeli regresyjnych.

Z kolei w pozycji [12] autorzy zaprezentowali analityczny sposób syntetycznego generowania wskazań magnitudy akcelerometru na postawie informacji o pozycji poszczególnych stawów animacji szkieletowej. W pracy [13] badacze zaproponowali użycie symulatorów IMU do generowania danych syntetycznych na podstawie nagrań wideo zgromadzonych na platformie YouTube. Zastosowana Aplikacja *IMUSim* [14] pozwoliła na odtworzenie serii czasowych wartości pomiarowych trójosiowych akcelerometrów, żyroskopów i magnetometrów na podstawie informacji o orientacji czujnika. Oprogramowanie to jest w stanie w sposób analityczny odtworzyć przebiegi czasowe, jak i zasymulować typowe zakłócenia poszczególnych czujników.

Czerpiąc inspirację z publikacji [13] zdecydowano się na odtworzenie brakującego sygnału z wykorzystaniem metod analitycznych, odpowiadających stosowanym w symulatorach IMU. Podejście takie nie wymagało nauki modeli regresyjnych, a więc wcześniejszej znajomości sygnałów prędkości kątowej (które nie były udostępnione w przypadku omawianej bazy danych). Nadmienić należy, iż podejście takie nie było prezentowane wcześniej dla opisywanego korpusu danych.

2.1.5. Metody łączenia danych

W literaturze podano kilka odmiennych podejść do przetwarzania bloków danych pochodzących z różnych typów sensorów przez sieci splotowe CNN.

W pozycji [8] autorzy wykorzystywali nagrania zarejestrowane za pomocą akcelerometru oraz żyroskopu, zlokalizowanych na nodze do identyfikacji osób. Do klasyfikacji wykorzystano sieć splotową, na której wejście podawano bloki danych o wymiarach $8\times N$. Pierwszy z wymiarów wynikał z faktu wykorzystania odpowiednio sygnałów pomiarowych trójosiowych akcelerometru oraz żyroskopu oraz ich magnitud. Liczba N oznaczała liczbę próbek sygnału.

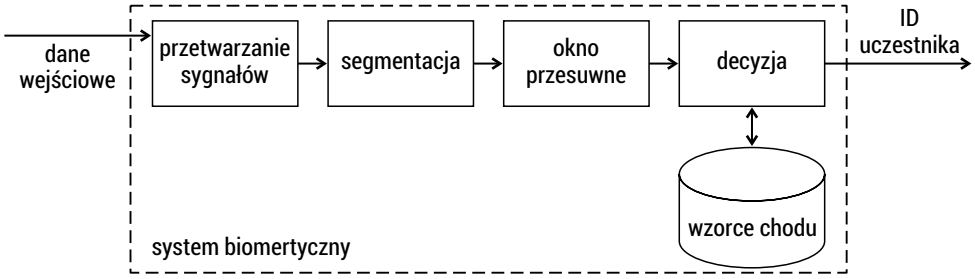
Z kolei w publikacji [5] autorzy wskazują, że mieszanie się nieprzetworzonych danych pomiarowych różnego typu (tj. pochodzących z sensorów mierzących odmienne wartości fizyczne) w pierwszej warstwie splotowej jest niepożądane. W pracy zaproponowano wstawianie bloków danych o zerowych wartościach (*tzw. Zero-padding technique*) pomiędzy zarejestrowane sygnały pomiarowe. Nadmienić należy, że wartości zerowe wstawiane były wyłącznie pomiędzy dane zarejestrowane przez sensory różnego typu. Przykładowo, pomiędzy sygnały pomiarowe dwóch akcelerometrów, nie umieszczano bloków wartości zerowych. Autorzy w publikacji argumentowali, iż prezentowane podejście doprowadzi do wyspecjalizowania filtrów splotowych do poszczególnych typów zarejestrowanych wielkości fizycznych.

Całkowicie odmienne podejście w zakresie łączenia danych z różnych typów czujników zaprezentowano w pozycji [9]. Autorzy przeprowadzili badania identyfikacji osób na podstawie wykonywanych przy pomocy dłoni w powietrzu symboli liter (ang. *Air-writing*). W swoich pracach zaproponowali wykorzystanie architektury sieci CNN o kilku wejściach (ang. *multi-input architecture*) do klasyfikacji danych zarejestrowanych z użyciem akcelerometru oraz żyroskopu. W prezentowanym podejściu dochodzi do wyspecjalizowania filtrów splotowych do poszczególnych typów danych. W przypadku architektury tego typu należy szczegółowo określić liczbę filtrów splotowych wykorzystywanych dla poszczególnych typów wartości pomiarowych.

2.2. Metodologia

Celem opracowanego systemu biometrycznego była identyfikacja osób na podstawie sygnałów pomiarowych czujników ruchu zarejestrowanych podczas chodu. Wejście systemu stanowiły sygnały ruchu oryginalnie znajdujące się w korpusie danych, wyjście zaś numery identyfikacyjne uczestników badań. Zaproponowany system posiadał następujące bloki: „*przetwarzanie sygnałów*", „*segmentacja*", „*okno przesuwne*" oraz „*decyzja*". Schemat blokowy rozwiązania zaprezentowano na rys. 2.1.

W bloku „*przetwarzanie sygnałów*" dokonywano przekształceń mających na celu zwielokrotnienie oryginalnych sygnałów ruchu. Proces ten miał na celu odnalezienie takiej formy reprezentacji danych, która zapewniłaby wysokie wskaźniki identyfikacji.



RYSUNEK 2.1. Schemat blokowy systemu biometrycznego

W etapie „*segmentacja*" dane poddane zostały procesowi prostej segmentacji. Powodował on usunięcie z nagrań fragmentów, w których uczestnik pozostawał w bezruchu. Proces ten umożliwił dalszą pracę na fragmentach rzeczywiście reprezentujących chód. W bloku „*okno przesuwne*" (ang. *sliding window*) poszczególne nagrania podzielone zostały na ramki danych o stałej długości. Umożliwiło to bezpośrednie podanie ich na wejście algorytmów klasyfikujących. Ponadto, zastosowanie okna przesuwnego z nakładkowaniem wynoszącym 50% zapewniło zwielokrotnienie danych treningowych. Finalnie w bloku „*decyzja*" z udziałem sztucznych głębokich sieci neuronowych typu CNN dokonano identyfikacji osób. Zaproponowane kroki przetwarzania omówione zostały szczegółowo w następnych sekcjach.

2.2.1. Przetwarzanie sygnałów

W prezentowanych eksperymentach zastosowano przetwarzanie sygnałów, które można podzielić na następujące grupy: przekształcenia w dziedzinie kwaternionów, implementacja techniki uzupełniania zerami (przyp. przegląd literatury) oraz filtracja częstotliwościowa.

Przekształcenia w dziedzinie kwaternionów

Wykorzystany w eksperymentach korpus danych w swojej oryginalnej formie nie zawierał informacji o prędkości kątowej. Obejmował on jednak sygnały pochodzące z: akcelerometru, magnetometru, orientacji w postaci kątów Eulera oraz znacznik czasowy. W pracach [13,14] autorzy wykorzystali informacje o orientacji w postaci kwaternionów do symulowania prędkości kątowych. W celu odtworzenia wartości pomiarowych żyroskopu z danych dostępnych w korpusie, niezbędne było dostarczenie informacji o orientacji sensora w postaci kwaternionu.

Zagadnienie to realizowane może być w dwojaki sposób: przez konwersję z reprezentacji kątów Eulera oraz przez analityczne wyznaczenie obrotu pomiędzy dwoma wektorami [15]. Konwersja z postaci kątów Eulera do postaci kwaternionu (jak i przekształcenie odwrotne) jest standardową praktyką, dzięki czemu posiada implementację w pakiecie Aerospace Toolbox środowiska MATLAB. Zagadnienie konwersji pomiędzy reprezentacjami nie zostanie więc szczegółowo opisane w tekście.

Drugi stosowany w literaturze [15] sposób wyznaczenia orientacji w postaci kwaternionu bazuje na wyznaczaniu kąta obrotu pomiędzy dwoma wektorami (1):

$$\begin{aligned} \boldsymbol{a} &= \frac{\left(a_x, a_y, a_z\right)}{\|\boldsymbol{a}\|} \\ \boldsymbol{b} &= \frac{\left(b_x, b_y, b_z\right)}{\|\boldsymbol{b}\|} \\ \alpha &= \operatorname{acos}\left(\langle \boldsymbol{a} \mid \boldsymbol{b} \rangle\right) \end{aligned}, \tag{1}$$

gdzie:
a, b – znormalizowane trójelementowe wektory;
$\langle a|b\rangle$ – iloczyn skalarny wektorów a i b;
α – kąt obrotu pomiędzy wektorami.

Oś obrotu wyznaczona jest na podstawie iloczynu wektorowego zgodnie z równaniem (2):

$$\boldsymbol{n} = \frac{a \cdot b}{\sin\left(\alpha\right)}, \tag{2}$$

gdzie:
$\boldsymbol{a}\times\boldsymbol{b}$ – iloczyn wektorowy wektorów a i b.

Finalnie kwaternion reprezentujący obrót pomiędzy wektorami wyznaczony jest zgodnie z równaniem (3):

$$\boldsymbol{q} = \left[cos\left(\alpha/2\right) \quad n_x \cdot sin\left(\alpha/2\right) \quad n_y \cdot sin\left(\alpha/2\right) \quad n_z \cdot sin\left(\alpha/2\right) \right], \tag{3}$$

gdzie**:**
$\boldsymbol{q}$ – kwaternion reprezentujący obrót pomiędzy dwoma wektorami;
n_x, n_y, n_z – poszczególne składowe wektora $\boldsymbol{n}$ (osie obrotu), wyznaczone w równaniu (2).

W przypadku przeprowadzanych badań wyznaczano orientację pomiędzy wskazaniami trójosiowego akcelerometru w chwili t, oraz globalnym wektorem orientację **G** – (0,0,1). Opisywany sposób pozwolił na wyznaczenie kwaternionu, reprezentującego orientację wyznaczoną jedynie na podstawie sensora akcelerometrycznego.

Seria czasowa kwaternionów reprezentujących orientację wraz ze znacznikiem czasowym posłużyć mogą do odtworzenia sygnałów prędkości kątowej. Proces ten rozpoczęty jest od wyznaczenia różniczki kwaternionu zgodnie z równaniem (4):

$$\dot{\boldsymbol{q}}_t = \left(q_{t+1} - q_t\right) / \Delta T, \tag{4}$$

gdzie:
$q_{(t+1)}$, q_t – orientacja w postaci kwaternionu w chwili *t+1* oraz *t*;
ΔT – różnica w czasie akwizycji próbki *t+1* oraz *t*;

Kolejnym krokiem jest odtworzenie prędkości kątowej zgodnie z równaniem (5):

$$\omega_t\left(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t\right) = 2 \cdot \boldsymbol{W}\left(\boldsymbol{q}_t\right) \cdot \dot{\boldsymbol{q}}_t\,, \tag{5}$$

gdzie:
ω_t – wektor prędkości kątowych (ω_x, ω_y, ω_z) w chwili *t*;
$\boldsymbol{W}$ – macierz mapująca kwaternion q_t oraz jego różniczkę, do prędkości kątowych wyrażonych we współrzędnych świata.

Wartość macierzy $\boldsymbol{W}$ zależy od wartości kwaternionu $\boldsymbol{q}$ w chwili *t*. Związek ten opisany jest równaniem (6):

$$\boldsymbol{W}\left(\boldsymbol{q}_t\right) = \begin{bmatrix} -q_{\mathrm{x}} & q_{\mathrm{w}} & -q_{\mathrm{z}} & q_{\mathrm{y}} \\ -q_{\mathrm{y}} & q_{\mathrm{z}} & q_{\mathrm{w}} & -q_{\mathrm{x}} \\ -q_{\mathrm{z}} & -q_{\mathrm{y}} & q_{\mathrm{x}} & q_{\mathrm{w}} \end{bmatrix}, \tag{6}$$

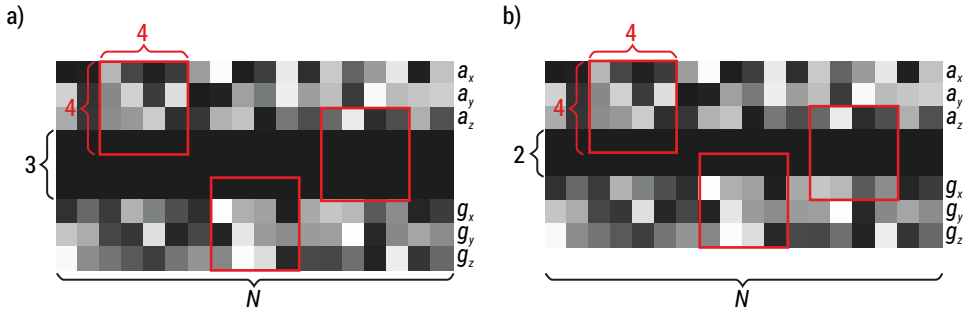
gdzie:
q_w, q_x, q_y, q_z – wartość składowych *w*, *x*, *y*, *z* kwaternionu w chwili *t*;

Dokonując właściwych eksperymentów skorzystano z własnoręcznie przygotowanej implementacji procedury wyznaczania prędkości kątowej. Skorzystano w tym celu z otwartych źródeł oprogramowania T-SIM [16]. Oryginalny kod źródłowy nie umożliwiał wyznaczenia prędkości kątowej dla nierównomiernie próbkowanego sygnału orientacji (co ma miejsce w wykorzystywanym korpusie danych).

Technika uzupełniania zerami

Idea jaka przyświeca stosowaniu techniki uzupełniania zerami (ang. *Zero-padding technique*) [5] jest oddzielenie danych pochodzących z sensorów mierzących odmienne wielkości fizyczne, tak aby nie doszło do ich mieszania w pierwszej warstwie splotowej sieci neuronowej CNN.

Podczas prac wykorzystano bloki danych o wartościach „0" oraz wymiarze 3×*N*, gdzie *N* oznacza długość ramki danych. Pierwszy z wymiarów wynika z faktu, że dane pomiarowe poszczególnych trójosiowych sensorów mają wymiar 3×*N*, a zastosowane filtry splotowe wymiar 4×*4*. Na rys. 2.2 zaprezentowano porównanie mieszania się danych dla bloków danych o wymiarze A) 3×*N*, oraz B) 2×*N*.



RYSUNEK 2.2. Technika uzupełniania zerami o wymiarze a) 3×N, b) 2×N, oraz filtach splotowych o wymiarach 4×4

Na osi *Y* rys. 2.2 zaznaczono: dane zarejestrowane z użyciem trójosiowego akcelerometru (tj. a_x, a_y, a_z), blok wartości zerowych oraz blok odtworzonych danych prędkości kątowych (g_x, g_y, g_z). Dla filtrów o wymiarze 4×4, w przypadku a) nie dochodzi do przetwarzania przez kernel danych z różnego typu sensorów. W przypadku b) dla ostatniego z kerneli nastąpi przetwarzanie z użyciem danych a_z oraz g_x. W przypadku stosowania trójosiowych sensorów wymiar wstawionych bloków danych powinien więc wynosić co najmniej 3 (dla filtrów splotowych o wymiarze 4×4).

Filtracja częstotliwościowa

W celu wyeliminowania szybkozmiennych zakłóceń (szum biały) w sygnałach pomiarowych akcelerometru [17] zastosowano cyfrową filtrację dolnoprzepustową. Zaimplementowano filtr Butterwortha czwartego rzędu o częstotliwości odcięcia wynoszącej 10 Hz. Podkreślić należy, że wartość ta dobrana była eksperymentalnie i ograniczona była częstotliwością Nyquista.

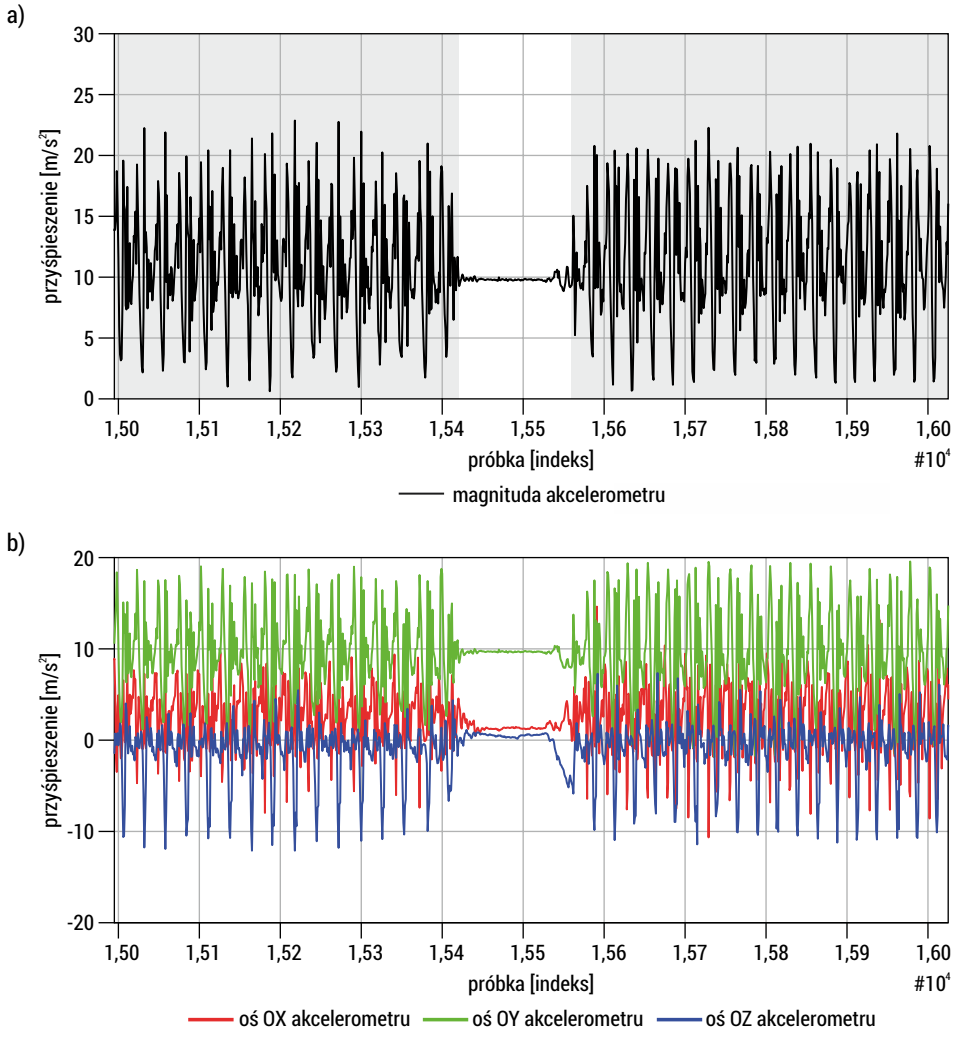
2.2.2. Segmentacja danych

Proces segmentacji danych może zachodzić w dwojaki sposób. Z jednej strony polegać może na wyselekcjonowaniu pojedynczych cykli chodu [3]. Z drugiej zaś strony dotyczyć może rozróżnienia chodu od sytuacji statycznych [18].

W pracach prezentowanych w rozdziale zastosowano drugą z omawianych metod. Proces wycinania fragmentów bezruchu, czy też małej aktywności, bazowano na inspekcji ręcznej. Zaznaczyć jednak należy, że kody źródłowe oprogramowania dostępne są na platformie Github [19]. Szczególnie istotne z punktu widzenia powtarzalności eksperymentów jest umieszczenie numerów próbek, które uwzględnione zostały w finalnych eksperymentach.

W literaturze popularne jest wykorzystanie informacji o magnitudzie sygnału akcelerometru w celu oceny aktywności uczestnika badań. O ile wskazania poszczególnych osi sensora zależą od jego orientacji [4,17], o tyle wskazania ich normy niosą jednoznaczną informację o ruchu czujnika.

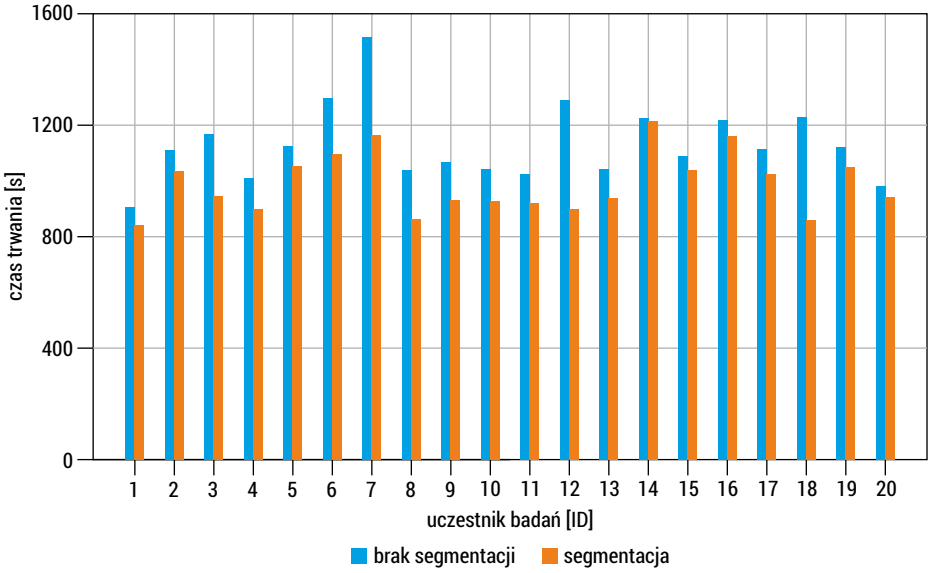
Na rys 2.3 zaprezentowano sygnały zarejestrowane podczas krótkotrwałej przerwy w chodzie. Na rys 2.3 a) wykreślono sygnał magnitudy akcelerometru. Za pomocą szarego tła zaznaczono fragmenty uznane za aktywny chód. Fragment sygnału o wartościach około 10 m/s^2 uznany został za przerwę w aktywności. Wartość ta wynika z rejestracji przyśpieszenia grawitacyjnego przez czujnik [17] i jest typowa dla sytuacji statycznych pozbawionych ruchu. Z kolei na rys 2.3 b) wykreślono sygnały zarejestrowane przez kolejne trzy osie akcelerometru. W przypadku przerwy w chodzie zauważyć można brak znaczących powtarzalnych zmian.



RYSUNEK 2.3. Sygnały zarejestrowane podczas krótkiej przerwy w chodzie: a) sygnał magnitudy akcelerometru; za pomocą szarego tła zaznaczono okresy uznawane za aktywny chód b) sygnały zarejestrowane przez poszczególne osie czujnika

Podkreślić należy, że w bazie danych przed zastosowaniem procesu segmentacji czas zarejestrowanych sygnałów przez każdego uczestnika badań wynosił co najmniej 900 sekund (tj. 15 min). Jednakże, dla uczestnika badań o nr. identyfikacyjnym 7, długość zgromadzonych danych przekraczała 1500 sekund.

W wyniku przeprowadzonego procesu usuwania pauz w nagraniach dla poszczególnych uczestników zgromadzono co najmniej 800 sekund nagrań. Na rys. 2.4 zaznaczono liczebność nagrań z podziałem na poszczególnych uczestników. Dysproporcja w długości nagrań uczestnika #7 została zmniejszona.



RYSUNEK 2.4. Długość nagrań zgromadzonych dla poszczególnych uczestników badań przed oraz po zastosowaniu segmentacji

2.2.3. Podział danych na zbiór treningowy i testowy

Ewaluacja algorytmów klasyfikacji osób przeprowadzona została z wykorzystaniem prostej walidacji. Zbiór treningowy utworzony został na podstawie 60% czasu trwania nagrań, podczas gdy zbiór testowy stanowił 40% pozostałych danych. Zaznaczyć jednak należy, iż zarówno z początków, jak i końców poszczególnych nagrań, zostały usunięte bloki danych o długości 250 próbek (~8 sekund). Podejście takie wynikało z faktu, iż podczas ruszania, jak i hamowania, cykl chodu mógł być zaburzony.

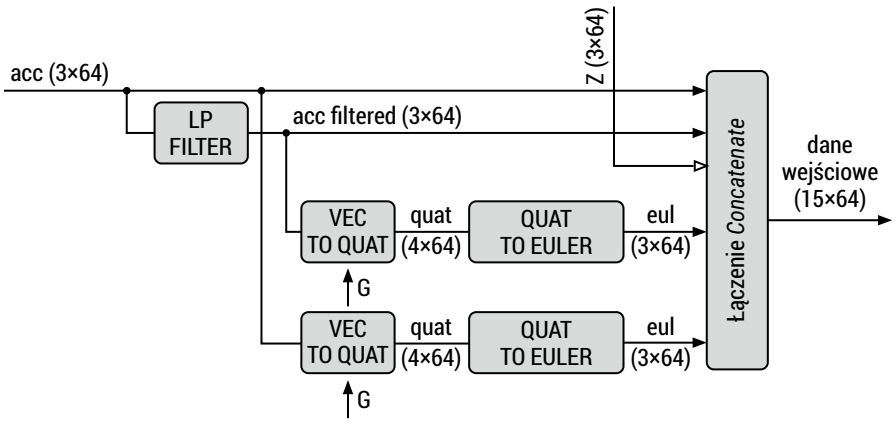
Podczas generowania zbiorów uczących oraz treningowych dane podzielone zostały na okna o czasie trwania 64 próbek oraz nakładkowaniu wynoszącym 50%. Przy częstotliwości próbkowania około 30 Hz długość okna wynosiła około 2,14 sekund, co jest zbliżone do wartości literaturowych, tj. 2,4 sekund [4] oraz 2,14 sekund [3].

2.2.4. Klasyfikacja chodu

Proces klasyfikacji chodu odbył się z wykorzystaniem splotowych sieci neuronowych przy wykorzystaniu biblioteki programistycznej PyTorch oraz języka programowania Python. Przeprowadzono szereg eksperymentów, w których dokonywano m.in selekcji danych wejściowych sieci, jak i zmian architektury sieci neuronowych. Wszystkie z opisywanych klasyfikatorów bazowały na architekturze typu AlexNet [21] i składały się z naprzemiennych warstw „*Convoluton*" oraz „*MaxPooling*". Do nauki sieci wykorzystano funkcję kosztu entropii krzyżowej (ang. *cross entropy*), optymalizator SGD (ang. *stochastic gradient descent*) oraz współczynnik nauki (ang. *learning rate*) o wartości 0,01.

System biometryczny I

W ramach pierwszego z eksperymentów sprawdzono możliwość podania na wejście sieci neuronowych dwóch typów sygnałów: przyspieszenia oraz orientacji. W tym przypadku dane wejściowe składały się z sygnałów przyspieszenia (w formie oryginalnej i filtrowanej) oraz sygnałów orientacji wyznaczonej na podstawie wartości pomiarowych przyspieszenia (w formie oryginalnej i filtrowanej). Poszczególne typy danych oddzielone zostały za pomocą bloków danych o wartości zerowej (o wymiarze 3×64). Przebieg łączenia danych wejściowych zaprezentowano na rys. 2.5. Orientacja wyznaczona została z wykorzystaniem wzorów (1), (2), (3). Wektorem ***G*** zaznaczono globalny wektor grawitacji.



RYSUNEK 2.5. Struktura danych wejściowych *Systemu I*

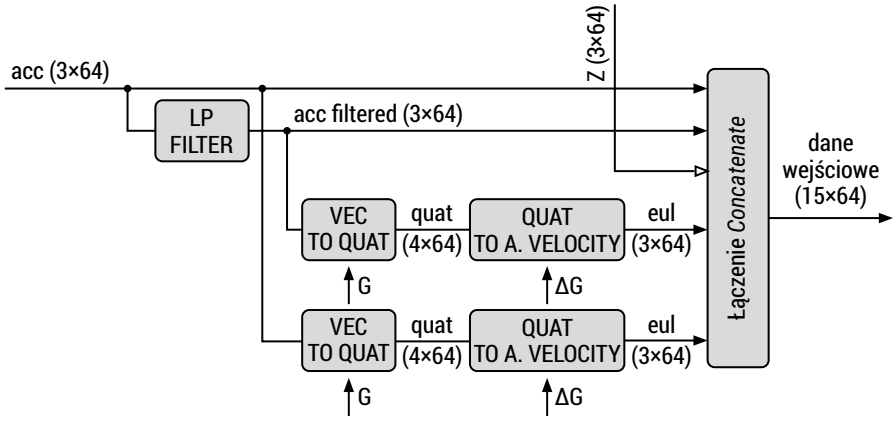
Szczegóły dotyczące sieci splotowej przedstawiono w tabeli 2.2. Zastosowanie sieci neuronowej w opisanej konfiguracji pozwoliło na osiągnięcie skuteczności klasyfikacji równej **51,4%**.

TABELA 2.2. Architektura *Systemu I* oraz *Systemu II, in/out – wejściowa/wyjściowa liczba filtrów, ks- kernel size, p-padding, s-stride*

Warstwy	Typ	Szczegóły
1	*Conv2D*	*in*=1, *out*=3, *ks*=(4,4), *p*=0
2	*MaxPool2d*	*ks*=(3,3), *s*=(1,1)
3	Conv2D	*in*=3, *out*=6, *ks*=(3,3), *p*=0
4	*MaxPool2d*	*ks*=(2,4), *s*=(1,1)
5	Conv2D	*in*=6, *out*=6, *ks*=(5,5), *p*=0
6	*MaxPool2d*	*ks*=(2,5), *s*=(1,2)
7	Conv2D	*in*=6, *out*=9, *ks*=(2,2), *p*=0
8	*MaxPool2d*	*ks*=(1,2), *s*=(1,2)
9	*Flatten*	
10	*Fully Connected*	99×20
11	*Dropout*	0,5
12	*Softmax*	

System biometryczny II

W przypadku kolejnego badanego modelu wykorzystano zmodyfikowany zestaw danych wejściowych. W tej konfiguracji bloki danych składały się z sygnałów przyspieszenia (w formie oryginalnej i filtrowanej) oraz sygnały prędkości kątowych odtworzonych na podstawie wartości pomiarowych przyspieszenia (w formie oryginalnej i filtrowanej). Przebieg łączenia danych wejściowych zaprezentowano na rys 2.6. Prędkość kątowa została wyznaczona z wykorzystaniem wzorów (4), (5), (6).

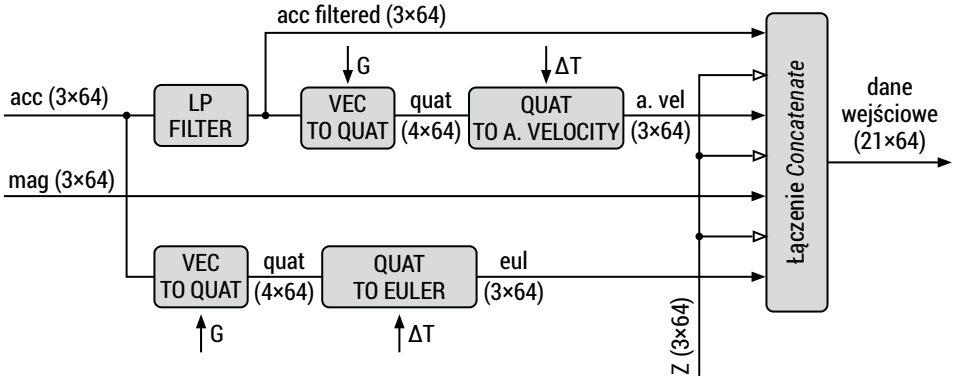


RYSUNEK 2.6. Struktura danych wejściowych *Systemu II*

Na wejście sztucznej sieci neuronowej podano dane o wymiarze 15×64. W porównaniu do *Modelu I*, w danych wejściowych zamiast sygnałów orientacji (w postaci kątów Eulera) znalazły się sygnały prędkości kątowej. Zastosowano dokładnie tę samą architekturę sieci, co w punkcie poprzednim. Zastosowanie sieci neuronowej w opisanej konfiguracji pozwoliło na osiągnięcie skuteczności klasyfikacji równej **54,9%**.

System biometryczny III

W drodze przeprowadzonych eksperymentów najlepszy wyniki klasyfikacji uzyskano przy wykorzystaniu danych wejściowych w postaci bloków składających się z sygnałów: filtrowanego przyspieszenia, prędkości kątowej, natężenia pola magnetycznego, oraz orientacji rys. 2.7.



RYSUNEK 2.7. Przebieg łączenia danych wejściowych dla *Systemu III*

Szczegóły dotyczące sieci splotowej przedstawiono w tabeli 2.3. Zastosowanie sieci neuronowej o takiej architekturze pozwoliło na osiągnięcie skuteczności identyfikacji równej **89,3%.**

TABELA 2.3. Architektura *Systemu III*

Warstwy	Typ	Szczegóły
1	*Conv2D*	*in*=1, *out*=2, *ks*=(4,4), *p*=0
2	*MaxPool2d*	*ks*=(3,6), *s*=(1,2)
3	Conv2D	*in*=3, *out*=6, *ks*=(3,3), *p*=0
4	*MaxPool2d*	*ks*=(2,4), *s*=(1,2)
5	Conv2D	*in*=6, *out*=6, *ks*=(5,5), *p*=0
6	*MaxPool2d*	*ks*=(2,5), *s*=(1,1)
7	*Flatten*	

Warstwy	Typ	Szczegóły
8	*Fully Connected*	192×40
9	*Sigmoid*	
10	*Fully Connected*	40×20
11	*Softmax*	

Zestawienie wyników eksperymentów

W tabeli 2.4 zaprezentowano zestawienie wyników badań dotyczących identyfikacji osób na postawie chodu. *System I* oraz *II* posiadały tożsamą strukturę sieci splotowej, a różnica w klasyfikacji wynikała z uwzględnienia różnego zestawu sygnałów wejściowych. Najlepsze wyniki identyfikacji wynoszące około 90% zaobserwowano dla ostatniego z systemów biometrycznych.

TABELA 2.4. Zestawienie skuteczności opracowanych systemów biometrycznych

System biometryczny	Skuteczność [%]
I	51,4
II	54,9
III	89,3

Podsumowanie

W niniejszej pracy przedstawiono wyniki eksperymentów dotyczących identyfikacji osób na podstawie chodu. W przeprowadzonych testach zbadano możliwość rozróżniania tożsamości na podstawie danych pomiarowych czujników ruchu.

Przyjęta metodologia badań zakładała zastosowanie prostej segmentacji, usuwającej z przebiegów jedynie okresy bezruchu. W prezentowanym podejściu brak jest algorytmów wyznaczania cykli chodu czy ich początków. Uzyskane wyniki identyfikacji osób (89,3%) zbliżone są wartościami do rezultatów stosowanych w literaturze [4], (87,5%), w której stosowano dane wejściowe w podobnej formie (tj. brak podziału na poszczególne cykle chodu). W przeciwieństwie do pozycji [3, 4], w toku których przetwarzanie wstępne danych oparto na generowaniu pseudo- obrazów, w niniejszej pracy zaproponowano bezpośrednie podanie sygnałów pomiarowych na wejście sieci splotowych CNN.

W przeprowadzonych badaniach dokonywano licznych eksperymentów z topologią sieci i strukturą danych wejściowych. Zastosowanie odtworzonych, a nie uwzględnionych w oryginalnym korpusie sygnałów prędkości kątowych, pozwoliło na zwiększenie skuteczności identyfikacji. Zaobserwowano wzrost o około 3,5 punktów procentowych dla *systemu II* względem *systemu I* (tabela 2.4). O ile wartość ta jest stosunkowo

niewielka, należy pamiętać, że o tyle wykorzystanie sygnałów prędkości kątowych pozwoliło na osiągnięcie skuteczności klasyfikacji równej 89,3% w finalnym systemie identyfikacji.

Zaznaczyć należy, że identyfikacja przeprowadzona została z wykorzystaniem danych zgromadzonych w ciągu jednego dnia. Prezentowane podejście charakteryzują pewne ułatwienia, np. każdy z uczestników badań posiada stały rodzaj obuwia. Naturalną kontynuacją rozpoczętych badań jest ich rozszerzenie tak, aby obejmowały one dane zgromadzone w ciągu kilku dni.

Badania zostały zrealizowane w ramach pracy nr WI/WI/1/2019, sfinansowanej ze środków Ministerstwa Nauki i Szkolnictwa Wyższego w Polsce.

Bibliografia

[1] Zou Q., Wang Y., Wang Q., Zhao Y., Li Q., *Deep Learning-Based Gait Recognition Using Smartphones in the Wild*, arXiv:1811.00338, 2020

[2] Wang S.; Liu J., *Biometrics on Mobile Phone* [in] *Recent Applications Biometrics*; InTech: Rijeka, Croatia, 2011

[3] Zhao Y., Zhou S., *Wearable device-based gait recognition using angle embedded gait dynamic images and a convolutional neural network*, Sensors, 2017, vol. 17, no. 3

[4] Zhong, Y., Deng Y., *Sensor orientation invariant mobile gait biometrics*, Proceedings of the IEEE International Joint Conference on Biometrics (IJCB 2014), Clearwater, FL, USA, 2014

[5] Ha S., Yun J.-M., Choi S., *Multi-modal convolutional neural networks for activity recognition,* IEEE International Conference on Systems, Man, and Cybernetics, 2015

[6] Dehzangi O., Taherisadr M., ChangalVala R., *IMU-based gait recognition using convolutional neural networks and multi-sensor fusion*, Sensors, 2017, vol. 17, no. 12

[7] Wong W. Y., Wong M. S., Lo K., *Clinical Applications of Sensors for Human Posture and Movement Analysis: A Review*, Prosthetics and orthotics international, 2017, 31

[8] Gadaleta M., Merelli L., Rossi M., *Human authentication from ankle motion data using convolutional neural networks,* Proc. IEEE Stat. Signal Process. Workshop, 2016

[9] Ding Y.,Xue Y., *A Deep Learning Approach to Writer Identification Using Inertial Sensor Data of Air-Handwriting*, IEICE Transactions on Information and Systems, 2019

[10] Banos O., Calatroni A., Damas M., et al., *Kinect=IMU? Learning MIMO signal mappings to automatically translate activity recognition systems across sensor modalities*, Proceedings of the International Symposium on Wearable Computers (ISWC), Newcastle, UK, 2012

[11] Rey V. F., Hevesi P., Kovalenko O., Lukowicz P., *Let there be IMU data: generating training data for wearable, motion sensor based activity recognition from monocular RGB videos.* Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers, ACM. 2019,

[12] Lago P, Takeda S., Okita T., Inoue S., *MEASURed: Evaluating Sensor-Based Activity Recognition Scenarios by Simulating Accelerometer Measures from Motion Capture* [in] Kawaguchi N. (ed.) *Human Activity Sensing Corpus and Applications*, Springer Series in Adaptive Environments, 2019

[13] Kwon H., Tong C., Haresamudram H., Gao Y.,et al., *IMUTube: Automatic extrac-tion of virtual on-body accelerometry from video for human activity recognition,*arXiv:2006.05675, 2020

[14] Young A. D., Ling M. J., Arvind D.K., *IMUSim: A Simulation Environment for Inertial Sensing AlgorithmDesign and Evaluation*, Proceedings of the 10th International Conference on Information Processing in Sensor Networks, Chicago, IL, USA, 2011

[15] Sawicki A., Walendziuk W., Idzkowski A., *The gravitational acceleration components elimination from the accelerometer measurement data*, Proc. SPIE 10031,2016

[16] Stanley M, *Simulate Motion for Sensor Fusion, materiały promocyjne firmy NXP*, https://www.nxp.com/company/blog/simulate-motion-for-sensor-fusion:BL-SIMULATE-MOTION-SENSOR-FUSION [dostęp: 1.12.2020]

[17] Bieda R., Grygiel, R., Galuszka, A., *Naive Kalman filtering for estimation of spatial object orientation, Methods and Models* in Automation and Robotics (MMAR), 20th International Conference, 2015

[18] Barralon P., Vuillerme N., Noury N., *Walk Detection With a Kinematic Sensor: Frequencyand Wavelet Comparison*, Proceedings of IEEE International Conference of Engineering in Medicine and Biology Society (EMBS), 2006

[19] Sawicki A. Repozytorium projektu Github https://github.com/asawicki-pb/gait-biometry [dostęp: 1.12.2020]

[20] Krizhevsky A., Sutskever I., Hinton G.E., *ImageNet classification with deep convolutional neural networks*. Commun. ACM 60(6), 2017

DOI: 10.24427/978-83-66391-84-0_3

Rozdział 3
Zagadnienia segmentacji tęczówki na potrzeby systemów biometrycznych

Mirosław Omieljanowicz

Streszczenie: Celem opracowania jest przedstawienie zagadnień związanych z wykorzystaniem obrazu tęczówki oka ludzkiego w biometrii. W przedstawianej pracy opisano, z jakich komponentów składa się system biometryczny oraz jakim przekształceniom podlega obraz tęczówki, aby mógł być wykorzystany do celów identyfikacji czy też autoryzacji ludzi.

Największą uwagę poświęcono procesowi segmentacji tęczówki. W sposób syntetyczny omówiono problemy związane z segmentacją obrazu. Przedstawiono także kilka najczęściej używanych algorytmów segmentacji. Jeden z nich został zaimplementowany w wersji podstawowej (powszechnie stosowanej), a w dalszej części opracowania wskazano i opisano kilka zmodyfikowanych zastosowań wybranego algorytmu. Przygotowane algorytmy zostały następnie poddane ewaluacji pod kątem poprawności segmentacji tęczówki oka.

W opracowaniu opisano metodę testowania oceny efektów działania różnych wersji procesu segmentacji. W dalszej kolejności pokazano badania przeprowadzone zgodnie z wybranymi metodami ewaluacji. Na zakończenie publikacji zaprezentowano uzyskane wyniki, dokonano ich krytycznej oceny i wyciągnięto wnioski dotyczące dalszych kierunków prac.

Słowa kluczowe: Biometria, Tęczówka, Segmentacja

Wprowadzenie

Biometria jest technologią pomiaru i analizy biologicznych cech człowieka, przede wszystkim w celu rozróżniania ludzi. Identyfikacja lub weryfikacja osób jest wykonywana przez pomiar powtarzalnych fizycznych lub behawioralnych cech charakterystycznych. W przeciągu wielu lat prac badawczych zaproponowano wiele technik opartych o różnorodne cechy ludzkie, m. in odcisk palca, geometrię dłoni, wygląd twarzy, kształt ucha, siatkówkę oka, głos, podpis odręczny, zapach ciała, tęczówkę, sposób chodzenia, układ żył na dłoni. Kluczowym elementem definiującym znaczenie danej cechy w rozpoznawaniu człowieka są jej wysoka unikalność, uniwersalność, trwałość

i mierzalność. Do takich cech ewidentnie zalicza się tęczówka oka, jednakże niezbyt prosty i łatwy sposób pozyskiwania jej obrazu oraz stosunkowo skomplikowany obliczeniowo proces analizy spowodował, że dopiero w obecnym okresie pojawiają się komercyjne rozwiązania dostępne dla przeciętnego użytkownika [1].

Tęczówka oka jest jedną z głównych biometryk stosowanych w zaawansowanych systemach dostępu do budynków i biur oraz będących tematem licznych publikacji naukowych i popularnonaukowych. Główną jej przewagą nad innymi podejściami jest olbrzymi poziom różnorodności wzorów jej wyglądu. Według źródeł literaturowych prawdopodobieństwo, że w procesie porównywania dwóch obrazów od dwóch różnych osób da błędny wynik wynosi jak 1 do 10^{12} [2,3]. Oznacza to, że wystąpienie fałszywej akceptacji jest w praktyce niemożliwe. Poza tym, tęczówka należy do najbardziej trwałych cech biometrycznych człowieka, gdyż nie ulega zmianie przez całe życie. Dodatkowo, przy bezpośredniej akwizycji tęczówki w czasie rzeczywistym, oszustwo z wykorzystaniem podstawionego wzoru (np. zdjęcia) jest praktycznie niemożliwe. Wymienione wyżej podstawowe cechy charakterystyczne tęczówki oka powodują, że w zastosowaniach biometrycznych jest ona bardzo atrakcyjnym elementem.

Wykorzystanie tęczówki jako cechy biometrycznej determinuje wiele problemów, których rozwiązanie decyduje o jego skuteczności. O ile ekstrakcja cech (utworzenie wektora cech) z dobrze wydzielonego i znormalizowanego obrazu tęczówki nie stanowi istotnego problemu, o tyle w dziedzinie opracowania efektywnej metody wyznaczenia obszaru, na którym znajduje się tęczówka, stale trwają prace badawcze i rozwojowe. Jest to jeden z pierwszych i jednocześnie kluczowy etap analizy tęczówki oka, zwany segmentacją. Precyzja tego procesu decyduje o poprawności działania całego systemu. Niepoprawne działanie algorytmu segmentacji może prowadzić do powstawania różniących się reprezentacji wzoru tej samej tęczówki. Bezpośrednio wpływa to na współczynnik fałszywego odrzucenia i w rezultacie na ocenę efektywności całego systemu [3].

W dalszej części publikacji największą uwagę poświęcono metodom i algorytmom segmentacji tęczówki oka na obrazach w bliskiej podczerwieni z powodów wskazanych w pkt. 3.1.3 oraz pkt. 3.2.3. Eksperymentalnie pokazano w jakim stopniu zastosowane metody oraz algorytmy wpływają na skuteczność selekcji tęczówki mierzoną za pomocą opracowanej obiektywnej metody testowania i oceny efektów. Do przeprowadzenia badań wykorzystane zostały publicznie dostępne zasoby zawierające zdjęcia oczu, wykonane przy użyciu kamer operujących w bliskiej podczerwieni [4].

3.1. Zagadnienia segmentacji tęczówki oka

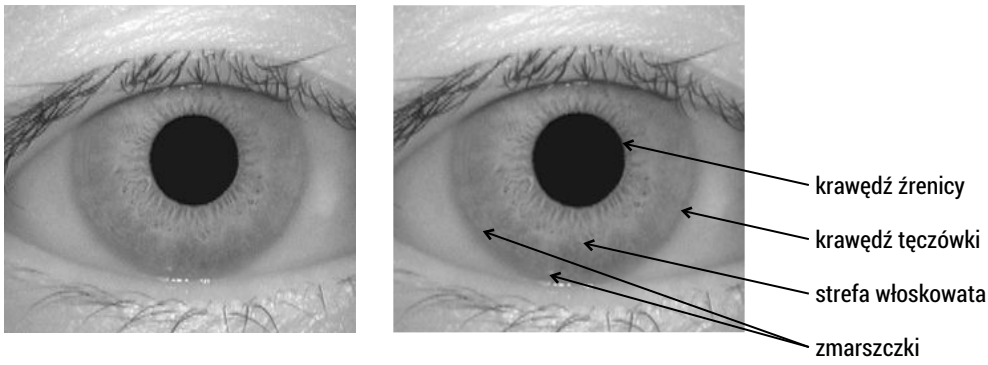
Segmentacja obrazu tęczówki oka należy do ogólnej dziedziny, jakim jest segmentacja obrazu, będącym typowym zagadnieniem z zakresu tzw. widzenia komputerowego. Polega to na wydzieleniu obszarów, które są jednorodne pod względem pewnych

własności, np. linii poziomych, pionowych, krawędzi obszarów o różnych czy też o tym samym kolorze, podobnym wzorze lub poziomie jaskrawości. W efekcie uzyskuje się uproszczenie lub zmianę prezentacji obrazu do formy, którą daje poszukiwany efekt końcowy lub jest przygotowana do dalszej analizy. Segmentacja obrazu jest głównie używana do ustalenia granic obszarów, na których znajdują się wybrane struktury czy obiekty. Inaczej ujmując, proces segmentacji polega na przypisaniu poszczególnych pikseli obrazu do pewnej klasy na podstawie wspólnych cech. Przykładowymi wynikami segmentacji mogą być zbiór konturów (pikseli na granicy dwóch obiektów), maska binarna (obraz z dwoma, zwykle skrajnymi poziomami wartości pikseli). Posegmentowany obraz często znajduje bezpośrednie wykorzystanie praktyczne w takich zastosowaniach, jak: widzenie maszynowe w automatyce, obrazowanie do celów medycznych, nadzorze wizyjnym, kontroli ruchu drogowego i oczywiście w biometrycznych systemach identyfikacji i autoryzacji ludzi.

Obecnie dostępnych jest bardzo wiele metod segmentacji, często bezpośrednio dostosowanych do ściśle ustalonego celu. Występują także metody uniwersalne. Część z nich wymaga współpracy z człowiekiem, część może działać w sposób nienadzorowany – automatyczny. Generalnie, metody segmentacji można podzielić na obszarowe i krawędziowe. Pierwszy z wymienionych typów opiera się na ustaleniu podobieństw wewnątrz określonego obszaru. W wyniku otrzymywane są zbiory pikseli podzielone na klasy według określonego kryterium. Drugi typ to metody krawędziowe służące ustaleniu granic oddzielających różniące się obszary wydzielając fragmenty składające się z punktów granicznych. Algorytmy stosowane w poszczególnych typach segmentacji są rozwiązaniami specjalizowanymi, tzn. algorytmy ustalające obszary nie pozwolą na wykrycie krawędzi i vice versa algorytmy krawędziowe nie sprawdzą się w identyfikacji obszarów. Przykładami powszechnie stosowanych metod segmentacji są progowanie, rozrost obszaru, różnego typu transformacje [5].

3.1.1. Specyfika obrazu tęczówki oka

Przedstawiana publikacja dotyczy analizy obrazu tęczówki oka. Oznacza to, że powinny być zastosowane specjalistyczne, dopasowane do specyfiki wyglądu oka rozwiązania. Tęczówka to część oka ludzkiego i nigdy nie jest widoczna samodzielnie, lecz jako pewien obszar na ujęciu oka. Tęczówkę z punktu widzenia anatomii tworzy kilka składników. Pierwszym z nich od wewnątrz organizmu człowieka, stanowi silnie ubarwiony nabłonek, nieprzenikliwy dla światła. Kolejnym elementem są mięśnie kontrolujące rozmiar źrenicy. Następny jest, silnie unaczyniony, tzw. zrąb tęczówki, który składa się w włókien kolagenowych. Najbardziej zewnętrzną warstwę stanowi nabłonek przedni. W nim położone są liczne komórki pigmentowe mające wpływ na kolor tęczówki. Jej wygląd obserwowany na zdjęciach jest efektem odbicia światła w jej wielowarstwowej strukturze [8]. Wygląd zewnętrzny ze wskazaniem charakterystycznych elementów przedstawiono na rys 3.1.



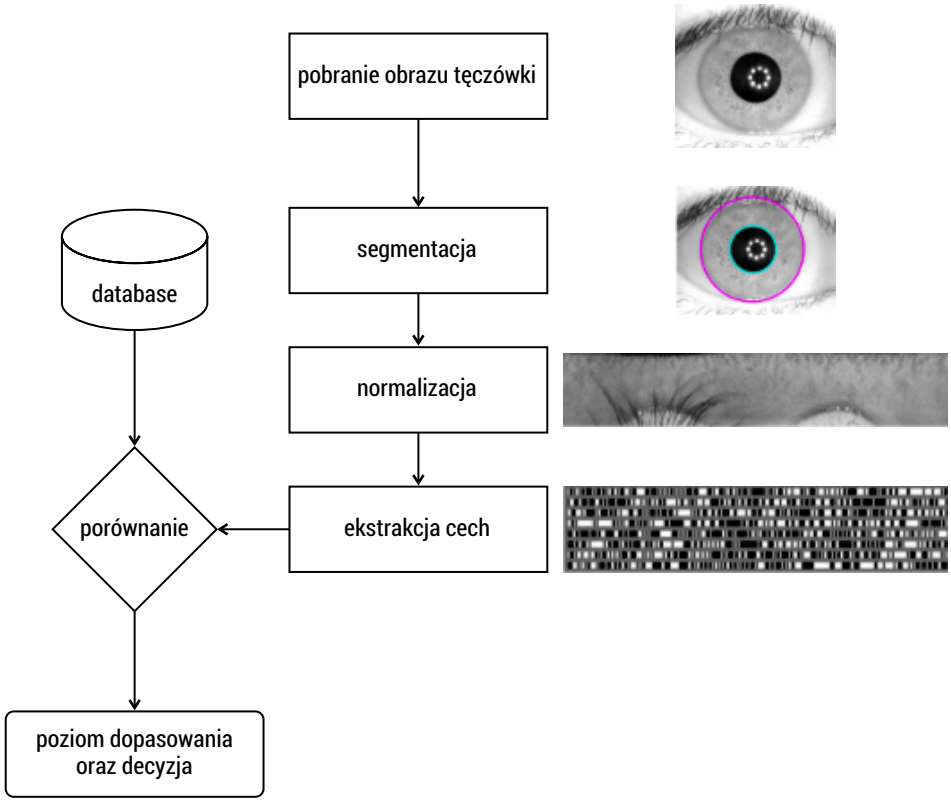
RYSUNEK 3.1. Obraz obu oczu z zaznaczonymi charakterystycznymi miejscami

Pokazane na rys. 3.1 zdjęcie oka zostało zrobione z bliska przy użyciu kamery w bliskiej podczerwieni (wyjaśnienie stosowania fal bliskiej podczerwieni, patrz pkt 3.1.3 i pkt 3.2.3). Widoczna na zdjęciu tęczówka ma kształt pierścienia ograniczonego od wewnątrz przez krawędź źrenicy i od zewnątrz przez granicę z twardówką. Wewnętrzy obszar można podzielić na dwie strefy. Pierwsza z nich od krawędzi źrenicy to strefa okołoźreniczna, zaś druga, otaczająca, to strefa włoskowata. Pomiędzy nimi znajduje się okrągłe zgrubienie nazywane kołnierzem. Wokół kołnierza widoczne są liczne zagłębienia i bruzdy. W pobliżu krawędzi tęczówki występują koncentryczne linie. Są to tzw. zmarszczki, które pogłębiają się przy rozszerzaniu się źrenicy. Opisane struktury tworzą, charakterystyczną dla tęczówki, teksturę i to ona jest elementem analizowanych w ramach systemów biometrycznych [6].

Obserwacje i analizy medyczne potwierdzają, że wygląd tęczówki jest unikalny dla każdego człowieka i od wieku dwóch lat nie zmienia się przez cały okres życia. Różnice występują nawet pomiędzy lewym i prawym okiem tej samej osoby, a także u bliźniąt jednojajowych. Według nauk medycznych wygląd zewnętrzny (tekstura na zdjęciu rys. 3.1) zależy do warunków, w jakich następował rozwój w fazie płodowej człowieka, a te są praktycznie niepowtarzalne u każdej osoby. Z tego powodu, mimo iż struktura tęczówki jest uwarunkowana genetycznie, to jest wręcz nieprawdopodobne, aby dwoje ludzi miało taki sam obraz/wzór tęczówki oka [8], a jest to najważniejsza cecha z punktu widzenia rozróżniania osób. Dodatkowo szerokość widocznego obrazu tęczówki zależy od rozmiaru średnicy źrenicy, której rozmiar zmienia się praktycznie w sposób ciągły. Obserwacja tych zmian może być użyta do weryfikacji czy analizowany jest obraz „na żywo." Zabezpiecza to przed podstawieniem sztucznego obrazu oka co oznacza wysokie bezpieczeństwo w relacji do prób oszukania [10] systemów autoryzacyjnych.

3.1.2. Ogólna struktura systemu rozpoznawania tęczówki

Każdy system biometryczny, w tym oparty o analizę obrazu tęczówki dokonuje porównania wektora cech bieżąco dostarczonej próbki z wektorami cech zgromadzonymi w bazie i ustala poziom podobieństwa wektora wejściowego ze wszystkimi w bazie poszukując najlepszego dopasowania a następnie decydując o zgodności próbki z wzorcem. Typowy blokowy schemat systemu rozpoznawania tęczówki wraz ze wskazaniem kierunku przepływu informacji został przedstawiony na rys. 2.2.



RYSUNEK 3.2. Schemat blokowy typowego systemu rozpoznawania tęczówki

Wyróżnione są cztery główne moduły realizujące następujące zadania (etapy przetwarzania): pozyskanie (akwizycja) obrazu, segmentacja tęczówki, normalizacja do postaci obrazu prostokątnego, ekstrakcja cech (wyznaczenie wektora cech) i porównanie z zapisami w bazie danych. Jakość wyników każdego z etapów jest bardzo uzależniona od efektów/poprawności działania poprzedniego.

Zadanie akwizycji obrazu polega na pozyskaniu dwuwymiarowego obrazu tęczówki. Obecnie zwykle wykorzystywane są do tego celu kamery CCD. W wielu sytuacjach stosowane jest dodatkowe oświetlenie źródłem podczerwonym w celu

uzyskanie poprawnego, odpowiednio jasnego zdjęcia. Proces akwizycji jest stosunkowo trudny w realizacji i wymaga aktywnego działania osoby poddającej się identyfikacji czy autoryzacji za pomocą obrazu tęczówki oka. W większości przypadków chodzi o odpowiednie otwarcie powiek i centralne ustawienie w okularze. Tak uzyskany obraz jest czasami wstępnie przetwarzany a potem przekazywany do modułu wykonania segmentacji, czyli „wycięcia" ze zdjęcia fragmentu zawierającego wyłącznie tęczówkę.

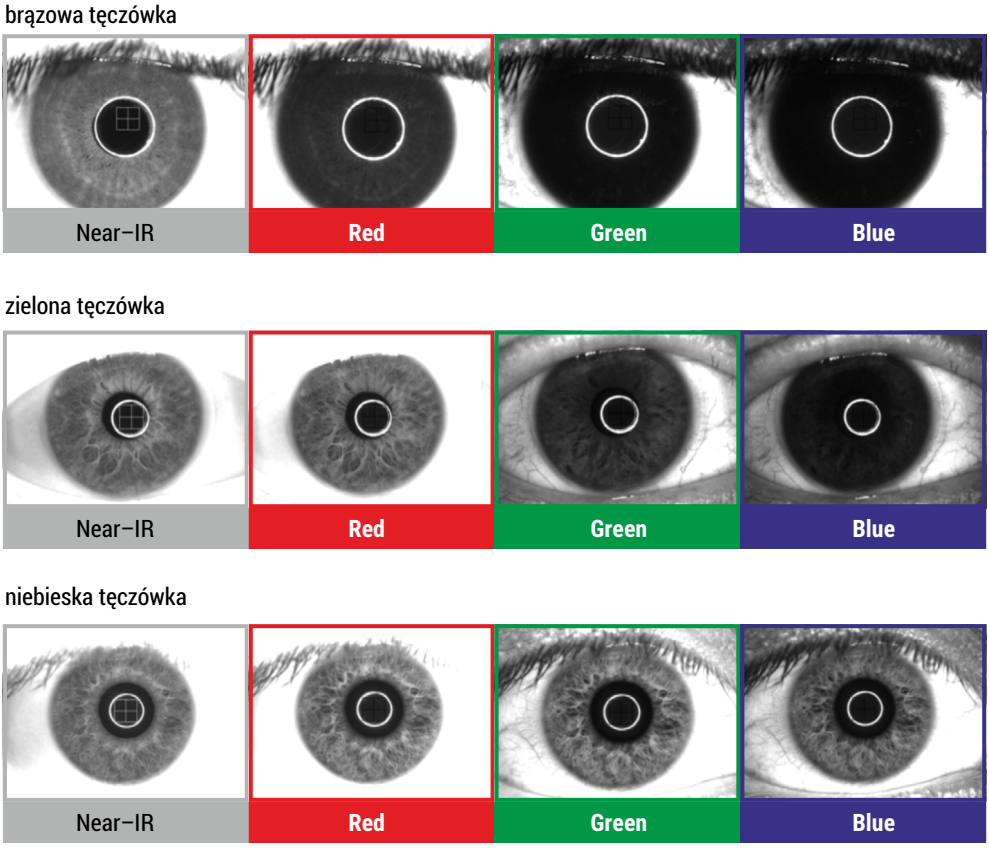
Blok segmentacji jest najistotniejszym elementem procesu analizy. Jego zadanie to lokalizacja obszaru zajmowanego przez tęczówkę i jego izolacja od pozostałych składników. Są to przede wszystkim źrenica, powieki, rzęsy, twardówka i różnego typu artefakty. Typowo segmentacja wykonywana jest w oparciu o wykrycie wewnętrznego i zewnętrznego konturu tęczówki i ewentualnie niepożądanych fragmentów powiek i rzęs, przysłaniających jej obszar. Ten proces jest krytycznym elementem systemu rozpoznawania tęczówki, a jego niedoskonałości drastycznie zmniejszają jakość (skuteczność) całego systemu biometrycznego.

Blok normalizacji przeprowadza geometryczne przekształcenie wysegmentowanego obszaru tęczówki mającego kształt pierścieniowaty do postaci prostokąta, ponieważ wykonanie obliczenia wektora cech na kształcie nieregularnego pierścienia jest bardzo skomplikowane. Znacznie prościej można to wykonać przy użyciu obrazu w typowej formie prostokąta. Operacja przekształcenia kształtu obrazu zwykle jest realizowana poprzez przeniesienie układu współrzędnych polarnych na układ kartezjański. Proces taki nazywany jest czasami „odwijaniem tęczówki" a jego rezultatem jest prostokątny pas, gdzie pionowa współrzędna odpowiada współrzędnej promieniowej, zaś pozioma współrzędnej kątowej. W tej operacji wytwarzany obraz jest sprowadzany do znormalizowanego kształtu o stałej ilości pikseli w pionie i poziomie. Takie podejście pozwala na uniezależnienie się od rozmiarów tęczówki wyodrębnionej w procesie segmentacji. Wymiary wydzielonej tęczówki różnią się bowiem u różnych osób, zmieniają się także w zależności od kąta ujęcia. Dodatkowo zwykle oprócz pożądanych składników wyodrębnionego obrazu zawiera on szereg zakłóceń, np. fragmenty rzęs czy powiek. Ich wyeliminowanie na pierwotnym kształcie (pierścieniowaty) jest trudne. Przejście na format znormalizowany pozwala na zastosowanie wielu ogólnie znanych metod poprawy obrazu. W procesie mapowania pierwotnego kształtu na znormalizowany przenoszone są także wszystkie artefakty i zakłócenia. Na prostokątnym obrazie można je eliminować przy użyciu ogólnie znanych metod i algorytmów, np. poprzez zastosowanie masek, filtrów. Oprócz tego, możliwe jest zastosowanie innych filtrów uwydatniających elementy tęczówki.

Kolejny blok to procedury wyznaczania wektora cech. Możliwe jest zastosowanie różnych podejść, które również wpływają na końcowy rezultat w postaci skuteczności systemu biometrycznego. Większość algorytmów wykorzystuje filtry falkowe, które z obrazu wyznaczają informacje o fazie [2,3,10]. W wyniku kwantyzacji fazy powstają wektory cech, które mogą być porównywane algorytmami wyznaczającymi odległości

3.1.3. Zagadnienia segmentacji tęczówki

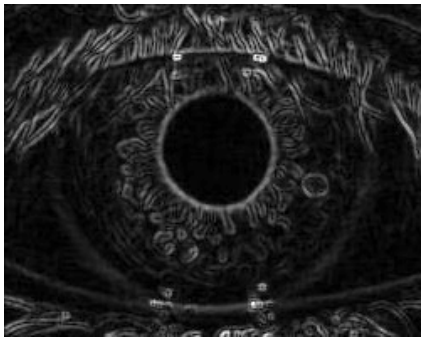
Ze względu na specyfikę fizycznej budowy tęczówki w rozwiązaniach biometrycznych najczęściej wykorzystuje się obrazy powstałe w ujęciach z zastosowaniem oświetlenia w bliskiej podczerwieni [9]. Cechy fizyczne tęczówki oka powodują, że szczegóły są słabo widoczne w widmie światła widzialnego. Wykorzystanie fal o większej długości, tuż poza zakresem widzialności znacząco lepiej uwydatnia strukturę tęczówki, dając tym więcej informacji potrzebnych wydzielenia odpowiedniego obszaru i budowy unikalnego wektora cech. Przyczyną takiego efektu jest pochłanianie przez melaninę części widma światła widzialnego. Zostało to pokazane na rys. 3.3. Obrazy z pierwszej od lewej kolumny to widok w bliskiej podczerwieni, pozostałe to składowe podstawowe światła widzialnego.



RYSUNEK 3.3. Tęczówka oka w bliskiej podczerwieni i świetle widzialnym [7]

Wykorzystanie fal w bliskiej podczerwieni ma jeszcze jedną ważną zaletę. Jest ono niewidoczne dla człowieka i nieszkodliwe w zakresie mocy, która pozwala na uzyskiwanie jasnych, wyraźnych obrazów.

Jednakże, mimo zastosowania oświetlenia dającego wysokiej jakości obrazy, segmentacja tęczówki należy do trudnych, złożonych zagadnień. Wynika, to z faktu obecności na zdjęciach nieusuwalnych w procesie jego tworzenia naturalnych elementów budowy oka, jak już wcześniej wymieniane źrenicę, twardówkę oka oraz powieki i rzęsy. Różnorodność uzyskiwanego obrazu jest na tyle duża, że do jej segmentacji należy użyć wielu metod analizy obrazu, w większości krawędziowych (obszarowe nie sprawdzają się na tak zróżnicowanych obrazach).



RYSUNEK 3.4 Krawędzie na obrazie tęczówki uzyskane z użyciem operatora Sobel'a w osi x i y

Główne problemy, które pojawiają się w procesie segmentacji tęczówki to jej kształt, który odbiega istotnie od powszechnie znanych figur geometrycznych. Wewnętrzna krawędź występuje na granicy ze źrenicą. I to jest najprostszy do wykrycia element, gdyż źrenica jest zasadniczo okrągła i przy dobrej jakości zdjęcia wyraźnie ciemniejsza od pozostałych części oka. Zdecydowanie trudnym zagadnieniem jest ustalenie zewnętrznej krawędzi, gdyż mimo iż pełny obraz oka pokazuje również postać okręgu to wyjątkowo rzadko jest on widoczny w praktyce. Najczęściej jest on przysłonięta powieką/powiekami czy też rzęsami. Powoduje to, że algorytmy służące wykrywaniu krawędzi okręgów dają bardzo niezadowalające rezultaty. Pojawiają się fałszywe nieregularne krawędzie prowadzące do sztucznego obniżenia lub powiększenia obszaru uznanego jako tęczówka. Dodatkowo cały proces utrudniają typowe problemy występujące w analizie obrazów, takie jak: nierównomierne oświetlenie, refleksy świetlne, zbyt niski czy zbyt wysoki kontrast, zbyt jasny lub zbyt ciemny obraz całości. W związku z powyższym w analizie służącej segmentacji tęczówki wykorzystuje się praktyce wiele algorytmów nakierowanych na wykrywanie krawędzi. Jednakże, doprowadzenie do wybrania odpowiedniego fragmentu wymaga wielu przekształceń, gdyż użycie jednego algorytmu nie daje oczekiwanych rezultatów. Poniżej przedstawiono obraz uzyskany przy wykorzystaniu filtru Sobela, pokazujący wiele nieużytecznych, wręcz przeszkadzających w segmentacji tęczówki krawędzi (rys. 3.4).

3.1.4. Podstawowe metody stosowane w segmentacji tęczówki

Lista dostępnych publikacji dotyczących metod segmentacji tęczówki jest bardzo obszerna. Zdecydowana większość z nich dotyczy metod krawędziowych [15], część zaś korzysta z rozwiązań opartych o metody aktywnych konturów, gdyż pozwalają uwzględnić nieregularną geometrię widocznego na zdjęciach obszaru tęczówki. Wśród najczęściej wykorzystywanych możemy znaleźć: klasyczny już operator całkująco-różniczkujący Daugman'a [2,3,10] oraz równie klasyczna transformacja Hough'a [11] oraz rzadziej spotykane aktywne kontury geodezyjne [12].

Operator całkująco-różniczkujący jest podstawową metodą stosowaną w rozwiązaniach komercyjnych o wysokiej skuteczności i jedną z najpopularniejszych metod segmentacji tęczówki wymienianych w publikacjach. Został on opracowany i opublikowany przez John'a Daugman'a w 1993 roku. Ze swej natury pracuje on jako detektor okrągłych krawędzi i używany jest do ustalania wewnętrznego i zewnętrznego konturu tęczówki na obrazie. W dalszej części procesu wykonywana jest zmiana poszukiwanej krzywej z okręgu na łuk (linie paraboliczne), co pozwala ograniczyć wpływ potencjalnego przesłonięcia przez fragmenty powieki. Operator Daugman'a umożliwia poszukiwanie i wykrycie krawędzi powiek, które może być prowadzone z wykorzystaniem różnych krzywych. Ustalenie położenia powiek daje istotne korzyści poprawiając precyzję wykrycia obszaru tęczówki. Szczegóły tego rozwiązania nie zostały opublikowane i nadal pozostaje duże pole do kolejnych prac.

Drugim często stosowanym w literaturze podejściem jest wykorzystanie transformacji Hough'a należącą do klasyki algorytmów analizy obrazów i służącą do znajdowania w obrazie geometrycznych kształtów takich jak proste czy okręgi. Korzystając z bazowej wersji transformaty opracowano wiele jej modyfikacji w zastosowaniach do segmentacji tęczówki [13]. Przed jej użyciem koniecznym jest przeprowadzenie procesu wykrywania krawędzi przeprowadzonego w wielu kierunkach. Na tak uzyskanym obrazie (mapa możliwie wszystkich krawędzi) użyta zostaje transformata Hough'a dająca w efekcie wskazanie okręgów otaczających tęczówkę. Możliwe jest także wykorzystanie tej transformacji Hough'a w celu wykrycia powiek. Ciekawe rozwiązanie pokazano w pracy [14], gdzie wykorzystano dwa oddzielne obrazy, krawędzi poziomych do ustalenia położenia powiek, krawędzi pionowych do ustalenia zewnętrznej granicy tęczówki, jednocześnie obniżając czas przetwarzania (oddzielne obrazy z mniejszą ilością obiektów są przekształcane szybciej niż jeden z dużą ilością szczegółów) i poprawiając precyzję segmentacji. Użycie transformacji Hough'a do wydzielenia tęczówki charakteryzuje się dwoma istotnymi i trudnymi do uniknięcia komplikacjami. Chodzi o dobór progu algorytmów wykrywania krawędzi oraz progu oceny ilości punktów w przestrzeni Hough'a Mają one kluczowy wpływ na jakość segmentacji i praktycznie niemożliwe jest ustalenie ich w sposób automatyczny. Dodatkowo transformacja Hough'a wymaga znaczących mocy obliczeniowych, co na pewno ogranicza jej zastosowanie w systemach pracujących w czasie rzeczywistym.

Aktualnie w dziedzinie segmentacji tęczówki oka bardzo obiecującym rozwiązaniem jest zastosowanie metody aktywnych konturów [12]. Opisane wyżej klasyczne podejścia zakładają, że krawędzie tęczówki mają kształt okręgu, co jest zgodne z anatomicznym obrazem. Jednakże, jak wskazano wyżej na zdjęciach ludzkiego oka (praktycznie w każdym przypadku) jest obserwowane przysłonięcie przez powieki ew. rzęsy. Implikuje to, że przy segmentacji konieczne jest wykrywanie krzywych niebędących częścią okręgu. W standardowym podejściu wymagane jest wtedy przeprowadzenie wielokrotnego wykrywania krawędzi w różnych kierunkach. W tym kontekście interesującą alternatywą są metody aktywnych konturów, które pozwalają na śledzenie i ustalanie nieregularnych krawędzi. Do segmentacji tęczówki zwykle jest stosowana ich odmiana geodezyjne aktywne kontury [12]. Proces ustalenia interesującego obszaru jest podzielony na dwa następujące po sobie generalne etapy – wydzielenie źrenicy i wydzielenie tęczówki. Segmentacja tęczówki jest uwarunkowana ustaleniem jej wewnętrznej krawędzi, co wskazuje, że w pierwszym etapie musi zostać wydzielona źrenica. Zastosowanie aktywnych konturów wymaga zainicjowania procesu segmentacji tęczówki okręgiem leżącym wewnątrz jej, dlatego w pierwszej kolejności wykonywana jest segmentacja źrenicy. Zostało to wykonywane stosunkowo prostym postępowaniem polegającym, w uproszczeniu, na ustaleniu minimalnej wartości piksela w obrazie (źrenica jest zwykle najciemniejszym elementem) i dokonanie binaryzacji obrazu z progiem wyższym o 25 od minimalnej wartości piksela. W efekcie otrzymuje się obraz zbinaryzowany zawierający tło i kształty odpowiadające źrenicy i innych elementów takich jak, np. rzęsy. Najmniejsze z elementów są usuwane filtrem medianowym a spośród pozostałych ustala się obszary możliwe do zakwalifikowania jako okręgi i wybiera największy jako obszar źrenicy. W następnym etapie wykorzystuje się algorytm aktywnych konturów geodezyjnych.

Metoda aktywnych konturów geodezyjnych bazuje na relacji pomiędzy aktywnymi konturami i obliczaniem linii geodezyjnych. Wyjściowym aktywnym konturem jest wyznaczona wcześniej linia wewnętrznej krawędzi tęczówki. Drogą ewolucji tej krzywej osiągana jest dość dokładna (względem wcześniej opisanych metod) segmentacja tęczówki. Jednakże, mimo dużej dokładności zastosowanie tej metody w praktyce wiąże się z kilkoma istotnymi problemami. Analogicznie do rozwiązań stosujących transformatę Hough'a bardzo ważny jest dobór parametrów (źle dobrane parametry skutkują słabej jakości segmentacją) i znacząca złożoność obliczeniowa na pewno ograniczająca jej wykorzystanie w praktyce.

Przedstawiona w ptk. 2 skondensowana analiza metod segmentacji tęczówki wskazała, że mimo upływu czasu najbardziej klasyczne rozwiązanie jakim jest zastosowanie operatora całkująco-różniczkującego Daugman'a ma przeważającą ilość zalet w stosunku do jej wad przede wszystkim, jeżeli ma się na celu praktyczne zastosowania. Decydują o tym takie cechy jak złożoność obliczeniowa pozwalająca na wykorzystanie jej w samodzielnie pracujących systemach wbudowanych i duże możliwości modyfikacji dające szansę na własne opracowania. W związku z powyższym

w dalszej części opisywanych w tej publikacji prac wykonano wdrożenie operatora całkująco-różniczkującego dokonując kilku zmian i uzupełniając szeregiem dodatkowych metod a także oceniono w sposób uporządkowany ich wyniki.

3.2. Badanie wariantów rozwiązań bazujących na algorytmie Daugman'a

3.2.1. Wykorzystane oprogramowanie

Zgodnie z rozważaniami przedstawionymi w poprzedniej części niniejszego opracowania do przeprowadzenia analizy różnych wariantów metod segmentacji tęczówki wybrano realizację opartą o algorytm Daugman'a. Jego idea i dostępne opisy pozwalają na wprowadzenie szeregu rozwiązań umożliwiających elastyczne zmiany „wokół" tego algorytmu oraz przeprowadzenie oceny ich wpływu na wysegmentowany obraz. Znajduje to potwierdzenie w ilości publikacji na jego temat, a także szeregu komercyjnych implementacji. W celu przeprowadzenia badań i eksperymentów wykorzystano oprogramowanie przygotowane w ramach pracy magisterskiej [18] na Wydziale Informatyki Politechniki Białostockiej. Zostało ona zaimplementowane w języku C++ z zachowaniem dużej modularności i konfigurowalności, co umożliwia użycie wielu wariantów rozwiązań i dość prostą modyfikację parametrów przetwarzania. Konstrukcja aplikacji przygotowana została z podziałem całego procesu na 5 podstawowych etapów, przy czym w każdym z nich umożliwiono łatwe wprowadzanie modyfikacji wybranych elementów. Etap 1 to przetwarzanie wstępne. W jego ramach możliwe jest zastosowanie algorytmów usuwania zakłóceń i poprawy kontrastu. Najważniejsze z nich to usuwanie zakłóceń poprzez zastosowanie jednego z filtrów (filtr Gaussa parametrze σ, filtr medianowy, filtr bilateralny), poprawa kontrastu (wyrównywanie histogramu lub adaptacyjne wyrównywanie histogramu z limitem kontrastu algorytmem CLAHE) czy też usuwanie refleksów świetlnych (metoda interpolacji obrazu tzw. *inpainting*). Drugi etap to wykrywanie krawędzi źrenicy (wewnętrznej krawędzi tęczówki). Zastosowany został tu operator całkująco-różniczkujący przystosowany do pracy na obrazach wykonanych w bliskiej podczerwieni. Możliwe jest też ustawianie poziomu głównego parametru jakim jest zakres promienia r wewnętrznego okręgu oraz postęp jego zmian (jest to jednocześnie krok operacji różniczkowania). Etap trzeci to wykrywanie zewnętrznej krawędzi tęczówki, również z wykorzystaniem operatora całkująco-różniczkującego, ale nie dla okręgów, lecz dla dwóch łuków lewej i prawej strony. Etap czwarty to wykrywanie górnej i dolnej powieki. Tu ponownie zastosowano operator całkująco-różniczkujący dostosowany do wykrywania łuku. Etap piąty to wykrycie rzęs i innych artefaktów oraz minimalizacja ich (usunięcie) wpływu na wynik segmentacji. na powierzchni tęczówki poprzez zastosowanie własnego algorytmu z podwójnym progowaniem.

3.2.2. Metoda oceny jakości segmentacji tęczówki

Segmentacja tęczówki jest krytycznym składnikiem biometrycznego systemu rozpoznawania czy też autentykacji ludzi. Mimo to skuteczność wykonania tego procesu nie da się łatwo ocenić poprzez badanie efektów działania całego systemu. Przyczyna jest oczywista – wynik końcowy zależy także od jakości działania jego pozostałych modułów składowych. Skupienie uwagi na efektach segmentacji i wykonanie ich oceny wymaga zastosowania specyficznego podejścia. Najbardziej oczywistym rozwiązaniem jest wizualna ewaluacja wykonana przez ludzi, ekspertów dziedzinowych. Takie podejście ma jednak istotne wady. Przede wszystkim jest oceną subiektywną. Poza tym, sprawdzi się przy ewaluacji stosunkowo niewielkiej liczby obrazów. W sytuacji dużej ilości wyników staje się czasochłonna i z upływem czasu rośnie ryzyko popełnienia ludzkiego błędu. Te fakty oznaczają, że konieczne jest zastosowanie metody oceny wyrażonej w wartościach liczbowych możliwych do ustalenia metodami obliczeniowymi zaimplementowanymi w postaci programów komputerowych, które gwarantują obiektywizm ewaluacji.

Generalnie obiektywne metody ewaluacji segmentacji obrazów bazują na wyznaczeniu różnicy wyników w stosunku do przygotowanych wzorców. W przypadku segmentacji obrazów polega to na zastosowaniu obrazu porównawczego przygotowanego jednorazowo ręcznie przez eksperta. Po przeprowadzeniu przetwarzania obliczana jest miara pokrycia otrzymanego wyniku z obrazem porównawczym. Porównanie jest wykonywane na podstawie obliczonych wartości miar, takich jak odległość lub poziom podobieństwa. Są one powszechnie stosowane ze względu na szybkość, powtarzalność i łatwość wyznaczenia.

Algorytmy segmentacji tęczówki zasadniczo dokonują podziału pikseli obrazu na dwie klasy – należących do tęczówki i stanowiących tło. Pozwala to na traktowanie tych algorytmów jako klasyfikatorów binarnych [17] i na zastosowanie sposobów ewaluacji stworzonych dla tego typu rozwiązań. Stanowi to znaczne ułatwienie, gdyż dostępnych jest wiele sprawdzonych miar stworzonych do oceny klasyfikacji binarnej. W takiej sytuacji typowo definiuje się zbiór pozytywny pikseli należący do obiektu (tu do obszaru tęczówki) oraz zbiór negatywny pikseli nienależących do obiektu (tu poza obszarem tęczówki). Poprzez porównanie przynależności pikseli w obrazie wynikowym z przynależnością w obrazie porównawczym możemy wyliczyć cztery charakterystyczne wielkości [16]. Są to wielkości:

- TP – (ang. *true positive*) – liczba pikseli prawidłowo określonych jako należące do obszaru tęczówki
- TN – (ang. *true negative*) – liczba pikseli prawidłowo określonych jako nienależące do obszaru tęczówki
- FP – (ang. *false positive*) – liczba pikseli błędnie określonych jako należące do tęczówki – tzw. błąd pierwszego rodzaju
- FN – (ang. *false negative*) – liczba pikseli błędnie określonych jako nienależące do obszaru tęczówki – tzw. błąd drugiego rodzaju

Na podstawie w/w czterech wielkości wyznacza się typowe miary stosowane w ocenie jakości przetwarzania obrazów. Są nimi precyzja (P), czułość (R od ang. recall) i miara F1 [16].

Precyzja jest wyznaczana wg. formuły (1) i wskazuje jaka część pikseli uznana jako należącą do tęczówki, została prawidłowo wybrana. Czułość obliczana wg. wzoru (2) wskazuje, jaka część wszystkich pikseli należących do tęczówki, została prawidłowo wyznaczona.

$$P = \frac{TP}{TP + FP} \tag{1}$$

$$R = \frac{TP}{TP + FN} \tag{2}$$

W/w wielkości dają ocenę wyniku w postaci dwóch wielkości, ale czasami komplikują przeprowadzenie jednoznacznego porównania (np. nie jest jasne, który z aspektów jest istotniejszy w danym zastosowaniu). Bezpośrednie porównanie jest możliwe przy użyciu pojedynczych wielkości. Taką możliwość stwarza trzecia z wymienionych powyżej wielkości. Jest to miara oznaczana jako F1, którą wyznacza się na podstawie formuły (3). Wielkość F1 nazywana jest czasami współczynnikiem Sørensen-Dice'a i stanowi syntetyczną ocenę, będąc średnią harmoniczną precyzji i czułości a jej maksymalna wartość odpowiadająca idealnej segmentacji wynosi 1.

$$F1 = \frac{2PR}{P + R} = \frac{2TP}{2TP + FP + FN} \tag{3}$$

Miary opisane powyżej dotyczą oceny pojedynczego obrazu wynikowego. W praktyce oczekujemy, że opracowane algorytmy będą stosowane do przetworzenia dużej ilości obrazów co oznacza, że przytoczone wskaźniki odnoszące się do pojedynczego obrazu nie są odpowiednio miarodajne. Do całościowej oceny algorytmów konieczne jest wykonanie ewaluacji dla zestawu obrazów pokrywającego dużą ilość przypadków spotykanych w praktyce. W takiej sytuacji należy zastosować podejście statystyczne. Wówczas możliwe jest porównywanie i ocena jakości dla dużych zbiorów danych. Typowymi wskaźnikami oceny statystycznej są wartość oczekiwana (zwykle oznaczana symbolem μ) oraz odchylenie standardowe (zwykle oznaczana symbolem σ). W przypadku ewaluacji jakości działania, rozważanej w tej publikacji segmentacji tęczówki ważną informacją jest określenie ilości przypadków, które w znaczący sposób różnią się do wartości oczekiwanej. Zwykle mają one niewielki wpływ na wartość oczekiwaną, jednak mogą wskazać pojedyncze/kilkukrotne przypadki, gdy metoda wyraźnie zawodzi, co daje możliwość „wyłapania" takich sytuacji i przeprowadzenia ew. udoskonaleń algorytmów. W statystyce są one określane jako obserwacje odstające a ich pomiar zwykle polega na wskazaniu ilości takich sytuacji. Typowo należą do nich przypadki, w których obliczona miara różni się od wartości oczekiwanej o więcej niż 3 odchylenia standardowe. W dalszej części publikacji liczba obserwacji odstających będzie oznaczana przez *o*. Przyjęto także, że jakość segmentacji będzie

oceniana miarą F1 poprzez jej wartość oczekiwaną (oznaczenie *μF1*), odchylenie standardowe (oznaczenie *σF1*) oraz ilość przypadków odstających (oznaczenie *oF1*). Przy tak przyjętej ocenie najlepsze wyniki to, te które mają najwyższą wartość oczekiwaną, przy małym odchyleniu standardowym i stosunkowo (do liczebności zbioru) małej ilości obserwacji odstających. We wszystkich opisanych niżej eksperymentach zastosowano miarę F1 z wyznaczaniem wartości *μF1, σF1 oraz oF1.*

3.2.3. Przygotowanie danych testowych

Opisywane wyżej metody przetwarzania i oprogramowanie testujące zostały nakierowane na przekształcanie obrazów oka wykonanych w bliskiej podczerwieni. Główną przesłanką do użycia takich obrazów był fakt publicznej dostępności znaczącej ilości próbek. Utworzenie własnej bazy obrazów oka w liczebności przekraczającej 100 to zadanie praktycznie wymagające bardzo długiego okresu ich gromadzenia a dodatkowo obwarowane istotnymi ograniczeniami prawnymi zawartymi w ustawach dotyczących ochrony danych osobowych. Wykorzystanie publicznie dostępnych zbiorów usuwa w/w problemy a jednocześnie daje możliwości stosunkowo prostego porównania własnych wyników z uzyskiwanymi przez innych wykorzystujących te same zestawy próbek. Wybrana została baza *CASIA Iris Image Database* [4], którą zbudowała Chinese Academy of Sciences – Institute of Automation, właśnie do celów związanych z systemami biometrycznymi. Zestaw obrazów w wersji 4 (CASIA-IrisV4) posiada 6 różnych zbiorów. Ze względu na to, że przygotowane metody i ich implementacja przygotowane zostały do badania skuteczności usuwania wpływu pewnych typowych jak i najtrudniejszych rodzajów zakłóceń, wyselekcjonowano 3 z nich w taki sposób, aby były w nich obrazy zawierające interesujące zniekształcenia. Wybrane zbiory to:

- *CASIA-Iris-Thousand – ilość osób 1000, ilość obrazów ok 20000, główne cechy – okulary na części próbek, refleksy świetlne pojawiające się na źrenicy i okularach, widoczny szeroki obszar wokół gałki ocznej łącznie z brwiami.*
- *CASIA-Iris-Lamp – ilość osób 411, ilość obrazów ok 16000, główne cechy – zmienny rozmiar źrenicy, nierównomierne oświetlenie.*
- *CASIA-Iris-Interval – ilość osób 249, ilość obrazów ok 2600, główne cechy –punktowe refleksy świetlne w obszarze źrenicy, obrazy prześwietlone, obrazy ze słabym kontrastem.*

*Zgodnie z opisem sposobu oceny jakości do jej przeprowadzenia konieczne są obrazy porównawcze przygotowane przez człowieka. Ze względu na czasochłonność ręcznego przeprowadzenia segmentacji z każdego ze zbiorów wybrano losowo p*o 200 próbek. Tak utworzone *zestawy zdjęć nazwano: zbiór Z1 – zawiera próbki z CASIA-Iris-Thousand, zbiór Z2 – próbki z CASIA-Iris-Lamp, zbiór Z3 – próbki z CASIA-Iris-Interval. W sumie przygotowano 600 zdjęć porównawczych. Format ich opisu wykonano tak, aby był zgodny z opisem wyniku uzyskiwanym z aplikacji segmentującej. Dzięki temu możliwe było automatyczne porównywanie z obliczaniem wskaźników statystycznych.*

3.3. Wybrane eksperymenty i wnioski

Jak opisano w podpunkcie 3.1 przygotowane do prowadzenia badań oprogramowanie posiada bardzo szeroki zakres wyboru zastosowanych metod i konfigurowania ich parametrów. Jest ich na tyle dużo, że realizacja wszystkich możliwych wariantów przetwarzania, ich opis i analiza wyników temat na oddzielne obszerne opracowanie. W przedstawionej publikacji zostały krótko opisane trzy wybrane próby, które dobrze wskazują na możliwości zastosowanego oprogramowania.

3.3.1. Wpływ parametru filtru Gaussa i kroku różniczkowania w operatorze całkująco-różniczkującym

Pierwszy z opisywanych eksperymentów dotyczy wpływu filtracji wstępnej obrazu za pomocą filtru Gaussa (oznaczenie parametru σG) i doboru kroku różniczkowania w operatorze Dougman'a (oznaczenie parametru Δr). Filtr Gaussa należący do kategorii uśredniających likwiduje wysokoczęstotliwościowy szum, jednak w efekcie otrzymujemy wygładzony obraz utrudniając tym wykrywanie krawędzi. Z kolej wielkość parametru Δr decyduje o skuteczności wykrycia krawędzi, przy czym niskie wartości sprawdzają się w przy „ostrych" obrazach, zaś duże przy obrazach wygładzonych. Zaplanowany eksperyment służy obserwacji efektów segmentacji przy różnych kombinacjach wartości współczynnika filtru Gaussa σG i kroku zmian promienia okręgu Δr w operacji różniczkowania operatora całkująco-różniczkującego. Testy zostały przeprowadzone na 3 zbiorach wskazanych powyżej. Parametr Δr zmieniano w zakresie 2–24 z krokiem 2, zaś σG w zakresie 0–10 z krokiem 0,5. Uporządkowane względem wartości $\mu F1$ wyniki (10 najlepszych) zostały pokazane w kolejnych tabelach 3.1, 3.2 i 3.3 odpowiednio dla zbiorów Z1, Z2 i Z3.

TABELA 3.1. Wynik dla zbioru Z1 względem parametrów Δr i σG

Lp.	Δr	σG	$\mu F1$	$\sigma F1$	$oF1$
1	10	6,5	0,804	0,314	27
2	10	8	0,801	0,311	28
3	10	7	0,800	0,315	29
4	10	6	0,800	0,320	29
5	10	7,5	0,799	0,314	29
6	10	8,5	0,797	0,309	28
7	10	9	0,792	0,310	28
8	10	5,5	0,792	0,328	31
9	16	7	0,786	0,299	26
10	16	7,5	0,785	0,298	26

TABELA 3.2. Wynik dla zbioru Z2 względem parametrów *Δr* i *σG*

Lp.	*Δr*	*σG*	*μF1*	*σF1*	*oF1*
1	12	7,5	0,874	0,149	15
2	12	7	0,874	0,154	16
3	8	7	0,870	0,180	18
4	12	5	0,868	0,183	17
5	12	5,5	0,868	0,176	17
6	12	6,5	0,868	0,163	17
7	8	6	0,868	0,197	18
8	8	7,5	0,866	0,189	16
9	12	6	0,865	0,177	18
10	12	4,5	0,863	0,197	17

TABELA 3.3. Wynik dla zbioru Z3 – parametry *Δr* i *σG*

Lp.	*Δr*	*σG*	*μF1*	*σF1*	*oF1*
1	20	6,5	0,882	0,134	7
2	22	5,5	0,880	0,119	10
3	22	6,5	0,880	0,119	10
4	22	6	0,879	0,120	10
5	20	7	0,879	0,139	8
6	20	7,5	0,876	0,136	10
7	22	7	0,875	0,125	11
8	20	8	0,874	0,137	11
9	22	7,5	0,873	0,132	12
10	22	8	0,871	0,127	14

We wszystkich przedstawionych tabelach wartość oczekiwana miary jest dość wysoka (powyżej 0,78) co oznacza, że segmentacja zlokalizowała tęczówkę prawidłowo. Wyraźnie najsłabsze wyniki widoczne są w tabeli 3.1 i dotyczą one zbioru Z1. Ewaluacja ręczna – przegląd wzrokowy plików zbioru Z1 wskazał, że najbardziej prawdopodobną przyczyną była duża ilość mocnych refleksów świetlnych.

3.3.2. Zastosowanie algorytmu usuwania refleksów i ograniczenia obszaru poszukiwań

Analiza wyników uzyskanych w eksperymencie opisanym w punkcie 3.3.1 wskazała, że najgorsze wyniki wynikały z obecności na zdjęciach refleksów świetlnych położonych na obrazie okularów poza przestrzenią, na której widoczna była tęczówka. Zainspirowało to do przeprowadzenia analizy z włączeniem do procesu przetwarzania algorytmu usuwającego refleksy oraz dodatkowo wprowadzenia ograniczenia obszaru poszukiwania położenia środka okręgu wyznaczającego wewnętrzną granicę tęczówki.

Zastosowany do ustalenia obszaru poszukiwań algorytm jest dość prosty i korzysta z parametru będącego progiem wartości piksela odniesionego do średniej jasności obrazu (oznaczony jako *wpj* – względny próg jasności). Z kolei do usunięcia refleksów świetlnych zastosowano metodę *inpainting*'u [17], przy czym pierwszą operacją było usunięcie refleksów na oryginalnym obrazie. Zmieniała ona jednocześnie średnią wartość jasności (usunięte najjaśniejsze punkty) co miało wpływ na bezwzględne wartości progu *wpj*, co dodatkowo uzasadniało połączenie wybranych metod w jednym eksperymencie. Eksperyment został przeprowadzony przy zmiennych wartościach progu *wpj* w zakresie 0,05–1 z krokiem co 0,05. Analogicznie jak w punkcie 3.3.1, poniżej przedstawiono wyniki (10 najlepszych) uzyskane dla wszystkich zbiorów wskazanych w tej publikacji.

TABELA 3.4. Wyniki testu – filtr antyrefleksu i ograniczenie obszaru poszukiwań, zbiór Z1

Lp.	*wpj*	*μF1*	*σF1*	*oF1*
1	0,25	0,913	0,116	6
2	0,2	0,909	0,132	7
3	0,3	0,908	0,132	7
4	0,15	0,907	0,134	8
5	0,35	0,903	0,147	8
6	0,4	0,903	0,147	8
7	0,45	0,903	0,147	8
8	0,5	0,903	0,147	8
9	0,55	0,903	0,147	8
10	0,6	0,900	0,153	9

TABELA 3.5. Wyniki testu – filtr antyrefleksu i ograniczenie obszaru poszukiwań, zbiór Z2

Lp.	*wpj*	*μF1*	*σF1*	*oF1*
1	0,2	0,898	0,082	16
2	0,35	0,895	0,096	15
3	0,3	0,895	0,096	15
4	0,25	0,894	0,101	13
5	0,4	0,890	0,115	13
6	0,45	0,890	0,115	13
7	0,55	0,885	0,131	14
8	0,6	0,885	0,131	14
9	0,65	0,885	0,131	14
10	0,7	0,885	0,131	14

TABELA 3.6. Wyniki testu – filtr antyrefleksu i ograniczenie obszaru poszukiwań, zbiór Z3

Lp.	*wpj*	*μF1*	*σF1*	*oF1*
1	0,4	0,903	0,064	11
2	0,45	0,903	0,064	11
3	0,5	0,903	0,064	11
4	0,55	0,903	0,064	11
5	0,6	0,903	0,064	11
6	0,65	0,903	0,064	11
7	0,7	0,903	0,064	11
8	0,75	0,903	0,064	11
9	0,8	0,903	0,064	11
10	0,85	0,903	0,064	11

Jak widać w prezentowanych tabelach wyniki mierzone wartością oczekiwaną uległy poprawie we wszystkich zbiorach. Dodatkowo odchylenie standardowe zmniejszyło się bardzo wyraźnie. Świadczy to o bardzo dużym wpływie punktowych jasnych zakłóceń na skuteczność segmentacji. W każdym rozpatrywanym przypadku (także gdy na obrazie nie było refleksów świetlnych) zastosowanie algorytmu *inpainting*'u poprawiało wynik. Należy stąd wyciągnąć wniosek, że w zadaniach segmentacji tęczówki powinien on być stosowany z definicji. Szczególnej poprawie uległy wartości standardowego odchylenia. Wskazuje to na pozytywne efekty także w zakresie powtarzalności. Jednakże, w niektórych przypadkach uległa zwiększeniu liczba obserwacji odstających. Można to uznać jako skutek zawężenia odchylenia standardowego, ale w niektórych zastosowaniach, gdy unikanie pojawienia się istotnie słabych wyników jest kluczowe

może stanowić wadę. Generalnie należy stwierdzić, że uzyskano znaczącą poprawę i użyty zestaw przekształceń i algorytmów może stanowić punkt wyjścia do budowy wysokoskutecznego systemu biometrycznego wykorzystującego tęczówkę.

3.3.3. Wykorzystanie algorytmu poprawy kontrastu

Mechanizmy dokonujące podziału obrazu na interesujące elementy zawsze opierają się na ustaleniu różnic i granic pomiędzy nimi. Potrzebują one odpowiednio przygotowanego zdjęcia. Jednym z najważniejszych czynników pozwalający rozróżniać zawartość obrazu, także dla człowieka, jest odpowiedni kontrast. Wydaje się, że dotyczy to także przypadku segmentacji (wydzielenia) tęczówki. Należy jednak uwzględnić fakt, że zmiany kontrastu wpływają na poziom jasności a precyzyjniej na rozkład jasności pomiędzy poszczególnymi obszarami obrazu. Już wstępne eksperymenty pokazały, że zastosowanie klasycznego podejścia jakim jest wyrównanie histogramu nie przynosi oczekiwanych rezultatów w postaci lepszego wykrywania położenia tęczówki. W tej metodzie dzieje się tak przede wszystkim z powodu powiększenia widoczności nie tylko samego obiektu, ale i elementów pobocznych takich jak rzęsy, krawędzie powiek czy inne podobne w charakterze zakłócenia. Wynika to z sedna zasady działania algorytmu wyrównania histogramu, który to operuje jednorodnie na całym obrazie.

Z wymienionych wyżej powodów zdecydowano się na wykonanie całego zbioru testów przy wykorzystaniu adaptacyjnego wyrównania kontrastu algorytmem CLAHE. Tak jak w poprzednich eksperymentach wykorzystano zbiory Z1, Z2 i Z3 i dokonano oceny wyników przy zastosowaniu miary F1. Zastosowano parametr limitu wartości kontrastu (oznaczenie *lk* – limit kontrastu) zmieniany w zakresie 0,25–10,0 z krokiem 0,25. Ponownie tak jak w poprzednich badaniach najlepszych 10 wyników przedstawiono w tabelach 3.7, 3.8, 3.9.

TABELA 3.7. Wyniki testu zbiór Z1 – adaptacyjne wyrównywanie histogramu

Lp.	*lk*	*μF1*	*σF1*	*oF1*
1	0,25	0,952	0,077	3
2	5,75	0,950	0,051	7
3	0,5	0,950	0,080	4
4	1	0,948	0,085	5
5	2,75	0,948	0,085	5
6	2	0,948	0,085	5
7	1,5	0,948	0,086	5
8	0,75	0,947	0,085	5
9	2,25	0,947	0,085	5
10	2,5	0,947	0,085	5

TABELA 3.8. Wyniki testu zbiór Z2 – adaptacyjne wyrównywanie histogramu

Lp.	*lk*	*μF1*	*σF1*	*oF1*
1	0,25	0,946	0,057	7
2	1,25	0,942	0,072	11
3	0,5	0,941	0,082	7
4	1	0,941	0,073	11
5	1,75	0,941	0,074	9
6	1,5	0,941	0,074	9
7	2,5	0,940	0,083	7
8	2,25	0,939	0,085	7
9	0,75	0,939	0,087	7
10	2,75	0,938	0,087	7

TABELA 3.9. Wyniki testu zbiór Z3 – adaptacyjne wyrównywanie histogramu

Lp.	*lk*	*μF1*	*σF1*	*oF1*
1	0,25	0,946	0,057	7
2	1,25	0,942	0,072	11
3	0,5	0,941	0,082	7
4	1	0,941	0,073	11
5	1,75	0,941	0,074	9
6	1,5	0,941	0,074	9
7	2,5	0,940	0,083	7
8	2,25	0,939	0,085	7
9	0,75	0,939	0,087	7
10	2,75	0,938	0,087	7

W przedstawionych wyżej, tabela 3.7, 3.8, 3.9, wynikach zaobserwowano (jak spodziewano się) poprawę jakości segmentacji tęczówki (wzrost wartości *μF1* i spadek liczby *oF1*). Jednakże należy zwrócić uwagę, że najlepsze wyniki dla każdego z trzech zbiorów osiągnięto przy minimalnym zwiększeniu kontrastu. Istotne powiększanie kontrastowości obrazu prowadziło do pogorszenia rezultatów, co potwierdziła obserwacja wzrokowa, gdyż przy większym kontraście uwydatniane mocniej były elementy zakłócające, np. rzęsy, krawędzie powiek, tak jak miało to miejsce przy zwykłym (nieadaptacyjnym) wyrównaniu histogramu. Wykonane na tym etapie badania kilkukrotnie wskazały, że duży wpływ na jakość segmentacji tęczówki ma obecność na zdjęciach elementów powiek i rzęs i w opracowanych rozwiązaniach przygotowano dodatkowe operacje służące ograniczeniu tego wpływu.

Podsumowanie

Celem przedstawionych w punkcie 4 eksperymentów było wykazanie wpływu różnych sposobów przetwarzania obrazów oczu na skuteczność segmentacji tęczówki. Jako sposób określenia jakości została wykorzystana często używana do oceny efektów przetwarzania obrazów miara F1 [16]. Przekształcaniu zostały poddane 3 zbiory testowe zdjęć pochodzących z publicznej bazy o liczebności 200 próbek każdy, a do porównań i wyciągania wniosków użyto wartości statystycznych miary F1, takich jak wartość oczekiwana i odchylenie standardowe oraz ilość obserwacji odstających. Testy były przeprowadzone za pomocą specjalnie przygotowanego oprogramowania umożliwiającego wykorzystanie wielu nastawionych na wydzielanie obszaru tęczówki metod przekształcania wraz z możliwością modyfikacji ich parametrów. Aplikacja była także wyspecjalizowana do oceny jakości segmentacji tęczówki rozumianej jako pewien etap przetwarzania w systemie biometrycznym. Zaprezentowane eksperymenty stanowią jedynie małą próbkę możliwości, wielo-wariantowości oprogramowania a jednocześnie pokazują możliwości uzyskiwania poprawy wyników poprzez wprowadzenie nowych nieskomplikowanych algorytmów, modyfikacji znanych rozwiązań oraz dobór wartości ich parametrów. Już przytoczone tylko 3 eksperymenty pozwalają na ustalenie jakie metody wnoszą istotną poprawę jakości segmentacji tęczówki. Należy do nich wprowadzenie w ciągu przekształceń, algorytmu usuwania refleksów świetlnych i poprawa kontrastu metodą adaptacyjnego wyrównania CLAHE. Zastosowane w aplikacji rozwiązania i specjalnie opracowane modyfikacje znanych metod dają duże możliwości badań nad osiągnięciem wysokich współczynników jakości segmentacji tęczówki. Inne, nie pokazane w tej publikacji eksperymenty z metodami wykrywania powiek, rzęs i innych artefaktów wskazują na potencjał całości opracowanych rozwiązań do odszukania zestawu algorytmów i wartości ich parametrów, dających wyniki segmentacji o jakości pozwalającej na zbudowanie całego, wysoce skutecznego pełnego systemu biometrycznego opartego na wykorzystaniu tęczówki jako unikalnej cechy człowieka.

Publikacja zrealizowana w ramach pracy zespołowej WZ/WI-IIT/4/2020 pt. „Biometria i przetwarzanie sygnałów"

Bibliografia

[1] Strona firmowa https://payeye.com [dostęp styczeń 2021]

[2] Daugman J.D., *High confidence visual recognition of persons by a test of statistical independence,* IEEE Transactions on Pattern Analysis and Machine Intelligence, 1993, vol. 15, no. 11

[3] Daugman J.D., *How iris recognition works*, IEEE Transactions on Circuits and Systems for Video Technology, 2004, vol. 14, no. 1

[4] CASIA Iris Image DataBase version 4, http://biometrics.idealtest.org/findTotalDbByMode.do?mode=Iris, [dostęp listopad 2020]

[5] Gonzales R., Woods R., *Digital Images Processing,* Pearson/Prentice Hall, 2008

[6] Wildes R.P., *Iris Recognition: An Emerging Biometric Technology*, Proceedings of the IEEE, 1997, vol. 85, no. 9

[7] Jain A.K., Ross A.A., *Introduction to Biometrics*, Springer, 2011

[8] Adler, F. H., *Physiology of the Eye*, fifth ed., St. Louis, Mosby, 1970

[9] Ślot K., *Wybrane zagadnienia biometrii*, WKiŁ, Warszawa, 2008

[10] Daugman J.D., *New Methods in Iris Recognition*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2007, vol. 37, no. 5

[11] Hough P.V.C. *Method and means for recognizing complex patterns*, U.S. Patent 3,069,654, Dec. 18, 1962

[12] Goldenberg R., Kimmel R., Rivlin E., Rudzsky M., *Fast geodesic active contours***,** IEEE Transactions on Image Processing, 2001, vol. 10, no. 10

[13] Wildes R.P., Asmuth J.C., Green G.L.,Hsu S.C., Kolczynski R.J.,Matey J.R., McBride S.E., *A system for automated iris recognition*, Proceedings of 1994 IEEE Workshop on Applications of Computer Vision, 1994, pp 121–128

[14] Kong W.K., Zhang D., *Accurate iris segmentation based on novel reflection and eyelash detection model,* Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong 2001, pp 263–266

[15] Malina W., Śmiatacz M., *Cyfrowe przetwarzanie obrazów*, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2008

[16] Powers D. M. W., *Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation*, Journal of Machine Learning Technologies, 2011, vol. 2, no. 1, pp. 37–63

[17] Bertalmío, M., Sapiro, G., Caselles, V., Ballester, C., *Image Inpainting*, Proceedings of SIGGRAPH 2000, New Orleans, USA, 2000. pp. 417–424. doi:10.1145/344779.344972

[18] Wejroch T., *Segmentacja tęczówki oka na obrazach w bliskiej podczerwieni*, Białystok 2018.

DOI: 10.24427/978-83-66391-84-0_4

Rozdział 4
Tworzenie uogólnionej architektury wersji oprogramowania z użyciem podobieństw struktur programowych

Cezary Boldak

Streszczenie: Wersje oprogramowania powstają, aby spełnić różniące się między sobą wymagania użytkowników / klientów. Pomimo istnienia technik umożliwiających generowanie takich wersji z jednego rdzenia, bardzo często powstają one przez kopiowanie i niezależne modyfikowanie wielu wersji kodu. Prowadzi to do duplikacji kodu, tzw. klonów programowych, i jest niezmiernie trudne w zarządzaniu. Architektura takich produktów, nawet jeśli przemyślana i udokumentowana pierwotnie, często ulega degradacji i traci się nad nią kontrolę. Brak znajomości aktualnej i spójnej między wersjami architektury prowadzi do wielu problemów, dlatego istnieją techniki odtwarzania takiej architektury na podstawie kodu źródłowego. Bazują one m.in. na identyfikowaniu odpowiadających sobie modułów programowych w wersjach na podstawie podobieństwa kodu, jednak zmiany naniesione w kodzie pomiędzy wersjami mogą zakłócić odnalezienie takich modułów. Dodanie do takich technik analizy klonów, mogącej zidentyfikować tylko podobne fragmenty i większe struktury kodu, powinno zwiększyć jakość odtwarzanej, uogólnionej architektury. W Pracy zaproponowano metodę uwzględnienia analizy klonów w metodach odtwarzania uogólnionej architektury wersji oprogramowania.

Słowa kluczowe: architektura oprogramowania, klony programowe, wersje oprogramowania

Wprowadzenie

Oprogramowanie jest obecne praktycznie w każdym aspekcie otaczającego nas środowiska. Od dedykowanych komputerów „zawędrowało" nawet wgłąb ludzkiego ciała, i nie jest to z pewnością proces zakończony. Ciągle rosnące zapotrzebowanie na programy spowodowało presję na ich szybsze, i – co z tym często związane – tańsze, dostarczanie. Aby zaspokoić tę potrzebę, w sytuacji braku czasu oraz niedostatku wykwalifikowanych twórców oprogramowania, idzie się na skróty – nowe programy powstają przez modyfikację programów istniejących, o zbliżonych funkcjonalnościach, bardzo często przez zwykłe skopiowanie i zmianę ich kodu. Wkrótce nad kolejnymi wersjami nikt nie panuje – nie wiadomo co w nich jest, dokumentacji

brak lub jest ona dawno zdezaktualizowana wskutek zmian nanoszonych na pierwotną formę, pierwotny zespół wytwórczy dawno już odszedł z firmy. Jeśli jednak zachodzi potrzeba modyfikacji takich wersji, brak wiedzy o konstrukcji programu jest chyba pierwszą przeszkodą, którą trzeba przezwyciężyć. Istnieją oczywiście metody odtwarzania architektury oprogramowania, i choć nie zawsze dają w pełni satysfakcjonujące efekty, to jednak stanowią dużą pomoc, zwłaszcza kiedy są w dużej mierze zautomatyzowanie. Jednak kiedy istnieje wiele wersji programu, w których objawił się (ten sam ?) błąd i kiedy czas „goni", chcielibyśmy prawdopodobnie poprawić go we wszystkich wersjach. Dlatego pożądane byłoby całościowe potraktowanie wszystkich wersji, żeby odnaleźć analogiczne miejsca w nich wszystkich. Możliwości takie oferuje uogólniona architektura, która opisuje wiele wariantów oprogramowania, wskazując podobieństwa i różnice między nimi. Jej istnienie jest już dużo rzadsze niż indywidualnych architektur każdej z wersji, ale i tu zaproponowane zostały techniki jej rekonstrukcji. W opinii autora w zbyt małym stopniu wykorzystują one fakt, że wersje te powstały z jednego kodu i pomimo zmian podobieństwo na poziomie kodu źródłowego przetrwało prawdopodobnie pomiędzy wersjami. W prezentowanej pracy podjeto próbę zaproponowania połączenia analizy klonów programowych, dziedziny zajmującej się właśnie podobieństwami w kodzie źródłowym programu lub programów, z istniejącymi technikami odtwarzania uogólnionej architektury oprogramowania istniejącego w wielu wersjach na podstawie kodu źródłowego tych wersji. Oprócz drobnych modyfikacji tych technik zaproponowano też nowe koncepcje, zmieniające te techniki w większym zakresie.

W podrozdziale 4.1 przedstawiono architektury oprogramowania proponowane przez różnych autorów, w 4.2 wymieniono kilka technik odtwarzania architektury pojedynczych systemów na podstawie ich kodu źródłowego, w 4.3 omawiono koncepcję linii produkcyjnych oprogramowania (SPL), mających usystematyzować wytwarzanie wielu wersji produktów, a w 4.4 – istniejące techniki tworzenia uogólnionej architektury, stosowanej w SPL. Podrozdział 4.5 poświęcono analizie klonów, ze szczególnym uwzględnieniem klonów strukturalnych. Zasadniczą część pracy stanowi podrozdział 4.6: przedstawiono tu propozycje uwzględnienia analizy klonów przy odtwarzaniu uogólnionej architektury. Pracę zakończono podsumowaniem.

4.1. Architektura oprogramowania

Architektura oprogramowania – czym jest ? Każdy jego twórca ma tu pewną intuicję, ale o jednolitą, powszechnie akceptowaną definicję trudno. Można podejść do architektury oprogramowania od bardziej technicznej strony, wyliczając co się musi w niej znaleźć, na poziomie bardziej abstrakcyjnym lub bardziej konkretnym. Według [1] w projektowaniu oprogramowania poziom architekturalny to coś co wychodzi ponad algorytmy i użyte struktury danych, obejmując organizację i globalną strukturę kontroli, protokoły komunikacyjne, synchronizacyjne, dostępu do danych, …

Standard IEEE [2] definiuje architekturę jako fundamentalną organizację systemu wyrażoną poprzez jego komponenty, zależności pomiędzy nimi oraz między komponentami i ich otoczeniem oraz zasady kierujące projektowaniem i ewolucją systemu. Bazując na tej definicji RUP [3] określa architekturę jako zbiór decyzji nt.: organizacji systemu, wyboru jego podstawowych komponentów i oferowanych przez nie interfejsów oraz ich działania i współdziałania, kompozycji tych komponentów w większe struktury, stylu architekturalnego wg którego podejmowane są poprzednie decyzje. Model widoków „4+1" [4] precyzuje jakie artefakty muszą znaleźć się w architekturze oprogramowania w celu udokumentowania perspektywy różnych udziałowców (stakeholders).

Jednak takie techniczne podejście nie jest jedyne. Fawler [5] widzi architekturę bardziej jako konstrukcję społeczną – należą tu rzeczy nie z definicji, ale ponieważ najważniejsze osoby w projekcie sądzą, że powinny się w niej znaleźć. Ponieważ architektura to decyzje, które najlepiej podejmować na początku projektu, to wynika stąd, że architektura to rzeczy, które ludzie postrzegają jako trudne do zmiany (dlatego podejmowane są na początku projektu). Przykładowo schemat bazy danych, determinujący większość kodu w aplikacji, będzie należał do architektury, ale jeśli zmiana takiego schematu jest łatwa i (najlepiej) automatyczna, to raczej będzie to detal a nie element architektury. W taki sposób, dążąc do łatwej zmiany wszystkich kluczowych elementów w projekcie, staramy się pozbawić go architektury ! Jednak jeśli jest mowa o automatycznym konstruowaniu architektury na podstawie kodu źródłowego bardziej właściwe będą tu techniczne definicje architektury określające je jako zbiór komponentów, czasem hierarchicznie uporządkowanych, współpracujących ze sobą i oferujących określone funkcjonalności (usługi) za pośrednictwem interfejsów.

4.2. Odtwarzanie architektury programowania na podstawie kodu źródłowego

W teorii powstanie architektury powinno poprzedzać wytworzenie innych artefaktów, w tym kodu źródłowego. Jednak jeśli nawet tak jest, to rzadko architektura jest uaktualniana w miarę zmian, którym podlega system. Nawet podczas implementacji mogą występować odstępstwa od architektury (czasem konieczne) powodujące rozbieżności między projektem a realizacją.

Istnienie aktualnej (!) architektury, nawet jeśli nie wymagane formalnie, ułatwia wiele zadań: rozbudowę, refaktoryzację, naprawę błędu, wprowadzanie nowych członków do zespołu opiekującego się systemem w fazie poprodukcyjnej. Wynika stąd potrzeba (re)konstrukcji architektury (lub przynajmniej jej elementów) na podstawie dostępnej informacji, którą może być kod źródłowy systemu.

Odtwarzanie architektury oprogramowania rozumiane jako grupowanie (klastering) podstawowych bloków (funkcje, moduły, ...) w większe komponenty może być wykonane przy użyciu wielu technik [6,7]: hierarchicznego grupowania [8],

algorytmów genetycznych [9] i grafowych [10], optymalizacji [11]. Źródłem danych do wyznaczenia architektury może być kod źródłowy i wszelkie statyczne informacje, które można z niego pozyskać (nazwy funkcji/metod, wzajemne wywoływanie, dostęp do danych, ...), ale także dynamiczne informacje uzyskane podczas wykonania zinstrumentowanych programów [12] czy też metainformacje: autorzy kodu czy umiejscowienie/nazwy plików [13].

4.3. Linie produkcyjne oprogramowania (SPL)

Systemy, które odniosły sukces, rzadko ograniczają się do jednej grupy użytkowników. Naturalne jest dążenie do ich adaptacji tak, aby mogły być użyte przez inne grupy użytkowników mających podobne, ale też różne wymagania względem systemu. Niestety, podobieństwo funkcjonalności nie oznacza ich identyczności, co zmusza twórców do opracowania nowych wersji, aby zastosować je wobec nowych wymagań. Nowe wersje mogą powstać na bazie oryginału, przez skopiowanie i zmodyfikowanie źródeł, co stanowić będzie znaczącą „pokusę" wobec znaczących oszczędności czasu i środków. „Pokusa" ta, niestety, „zemści" się prawdopodobnie w przyszłości, kiedy trzeba będzie zarządzać wieloma niezależnymi wersjami podobnego kodu – wszelkie zmiany, poprawki będą musiały być aplikowane do wielu linii bazowych (*baseline*) kodu.

W zorganizowaniu systematycznego procesu produkcji oprogramowania w wielu wersjach, zaproponowano koncepcję linii produkcyjnych oprogramowania (SPL – *software product lines*) [14]. Kolejne wersje systemów powstają poprzez łączenie gotowych komponentów – półproduktów (*assets*) pobranych z repozytorium. Każda wersja systemu może zawierać inny wybór takich komponentów, co jest pierwszym poziomem zmienności na najwyższym, architekturalnym poziomie. Wybór taki nie jest banalny, gdyż mogą pojawiać się zależności pomiędzy komponentami: komponenty wykluczające się, wymagane, alternatywne, ... Jednak taki poziom zmienności nie jest wystarczający, gdyż rzadko kiedy komponenty są na tyle generycznie napisane, aby spełniały wymagania wszystkich wersji produktu. Zwiększenie generyczności komponentu łączy się zwykle z jego większym skomplikowaniem w budowie i użyciu, mniejszą wydajnością. Dlatego tu też pojawią się kolejne wersje – tym razem komponentów – na niższym poziomie abstrakcji. Taką zmiennością komponentów też trzeba zarządzać, aby uniknąć powstania odrębnych wariantów po użyciu antywzorca: programowania w stylu *copy-paste-modify*. Użyć można wielu dostępnych technik: warunkowa kompilacja, metaprogramowania (szablony na wzór C++, klasy generyczne, ...), mechanizmy programowania obiektowego (dziedziczenie i kompozycja), wzorce projektowe [15], czy też dedykowanych rozwiązań typy język XVCL [16] (lub jego następna [17]). Wszystkie te wspomniane wyżej elementy (zestaw komponentów, ich wersje i wzajemne zależności, sposób doboru komponentów w określonej wersji w wariantach produktu) stanowią architekturę linii produkcyjnej oprogramowania, którą można nazwać architekturą uogólnioną, w odróżnieniu od konkretnej architektury określonego wariantu produktu.

4.4. Tworzenie uogólnionej architektury wariantów oprogramowania na podstawie wariantów produktów

Pomimo oczywistych zalet stosowania linii produkcyjnych oprogramowania, często powstaje wiele wariantów systemów informatycznych jako niezależnych produktów, czy to z powodu ich wieku, czy to nieświadomości twórców, czy to nieopłacalności opracowania SPL wobec niedoszacowanych planów rozwoju systemu [18]. Aby choć trochę ułatwić zarządzanie takimi wariantami, ewentualnie wspomóc wytworzenie linii produkcyjnej, wskazane jest opracowanie uogólnionej architektury na podstawie już istniejących wariantów produktów, zwykle mając do dyspozycji ich kody źródłowe [19].

PuLSE [20] używa adapterów (*wrappers*), aby przekształcić komponenty istniejących wariantów systemu w komponenty generyczne, możliwe do wielokrotnego użycia (*reusable*). Nie powstaje uogólniona architektura, ale takie podejście może być wykorzystane jako etap w procesie rekonstrukcji takiej architektury, zapewniając jej elementy składowe.

Metoda MAP (*Mining Architectures for Product Lines*), [21] proponuje procedurę (główne manualną) analizy architektur istniejących produktów w celu oceny, czy zawierają one potencjał do zbudowania z nich linii produkcyjnej oprogramowania. Potencjał ten jest rozpatrywany poprzez wskazanie elementów wspólnych (*commonalities*) i różnych (*variabilities*) na poziomie komponentów. Metoda rozpoczyna pracę od zbudowania modelu implementacyjnego (relację między jednostkami programowymi) każdej z wersji produktów oraz sformowanie (ręczne) komponentów (etap *bottom-up*). Następnie na taki model komponentów mapuje się (z użyciem wiedzy ekspertów) znane style architekturalne (etap *bottom-up*). Efekty pracy – niezależne architektury wersji produktów – są następnie używane do sformułowania wniosków na temat zasadności budowy SPL, jednak nie powstaje tu uogólniona architektura.

W ramach europejskiego projektu CAFÉ [22] uogólniona architektura (nazywana architekturą referencyjną – *reference architecture*) jest konstruowana przez analizę istniejących wariantów systemów, które mają być scalone w jednolitą linię produkcyjną. Jako etap wstępny wykonywana jest (re)konstrukcja architektur tych indywidualnych wariantów. Podobne podejście prezentowane jest w [23], gdzie na początku budowane są architektury pojedynczych członków rodziny oprogramowania sterującego robotami. Następnie budowany jest uogólniony model cech (*feature model*), który jest wzbogacany o możliwości adaptacji w celu zwiększenia ich uniwersalności. Na tej podstawie tworzona jest uogólniona architektura linii produkcyjnej, wraz z zestawem komponentów budujących pojedyncze warianty.

W pracach [24, 25] używa się uogólnionej metody refleksji Murphy'ego, aby porównać warianty oprogramowania na poziomie architektury. Oryginalna metoda refleksji działa dla pojedynczych programów, gdzie rzeczywiste elementy struktury programu mapowane są na hipotetyczny model architektury (przybliżony,

nieaktualny, ...) – w iteracyjnym procesie zmieniającym obydwie strony mapowania przybliżana jest rzeczywista architektura. Modyfikacja w celu zastosowania do SPL zakłada 2 procesy uzgadniania: implementacji z architekturą pojedynczego wariantu oraz architektury pojedynczego wariantu z uogólnioną architekturą SPL. Te 2 procesy wykonywane są sekwencyjnie dla każdego wariantu, przy czym mapowania w danym kroku musi uwzględniać (i ewentualnie modyfikować) poprzednie fazy. Po analizie ostatniego wariantu otrzymywana jest finalna uogólniona architektura SPL. Uogólniona architektura jest rozszerzeniem UML przez wprowadzenie komponentów obowiązkowych, opcjonalnych i wariantowych. W celu identyfikacji odpowiadających sobie jednostek programowanych (np. funkcji) pomiędzy wersjami (co jest potrzebne w uzgadnianiu mapowań pomiędzy krokami sekwencji) zastosowano analizę prostych klonów (podobieństw w kodzie źródłowym).

[26] proponuje technikę wydobywającą informację o zmienności w rodzinie produktów. Mapuje ona zbiór cech (*features*) i ich interakcji na artefakty programowe z założeniem, że zbiór tych cech jest znany. Relacje między cechami określone są przez ich model (graf). Mapowanie pozwala np. na wybór wariantu, który zawiera podzbiór cech najbardziej zbliżony do cech produktu, który powinien być wytworzony, aby proces tworzenia (za pomocą techniki *copy-paste-modify*) zacząć od takiego istniejącego wariantu.

Zaproponowana ostatnio metoda odtworzenia architektury SPL [27] używająca formalnej analizy konceptualnej (FCA – *formal concept analysis*) oferuje praktycznie automatyczne jej zbudowanie na podstawie kodu źródłowego wariantów. Tu również na początku odtwarzane są architektury poszczególnych wariantów. Następnie identyfikuje się występujące w nich komponenty w różnych wariantach. Odpowiedź na pytanie, czy użyte komponenty z poszczególnych architektur stanowią różne warianty jednego komponentu bazowego, uzyskuje się porównując serwisy przez nie oferowane: obowiązkowe (obecne we wszystkich wariantach) i opcjonalne (obecne jedynie w niektórych wariantach). Serwisy te reprezentowane są przez nazwy publicznych klas zawartych w komponencie. Komponenty z różnych architektur łączone są za pomocą hierarchicznego grupowania, używając kosinusowej miary podobieństwa (opartej na nazwach klas), produkując dendrogram. Węzły w dendrogramie spełniające określony warunek (podobieństwo elementów węzła przewyższa średnie podobieństwo w potomkach) wyznaczają komponenty bazowe, liście węzłów stanowią zaś komponenty-warianty komponentu bazowego. Dla komponentów bazowych określa się ich zmienności (*variabilities*): wewnętrzną (jako zbiór obowiązkowych i opcjonalnych klas) oraz zewnętrzną (jako zbiór obowiązkowych i opcjonalnych interfejsów, wymaganych i oferowanych przez komponent). Oferowane interfejsy w komponentach są reprezentowane przez serwisy (klasy) komponentów. Decyzja czy poszczególne interfejsy są uznawane za podobne (odpowiadające sobie) w wariantach komponentów zapada na podstawie podobieństwa kodu źródłowego implementujących je metod. Na końcu określa się komponenty obowiązkowe i opcjonalne w całej uogólnionej architekturze SPL. Dodatkowo określane są zależności pomiędzy komponentami: alternatywa, AND, OR, wymaganie, wykluczanie.

Niektóre warianty produktów mogą zbyt daleko odejść od pierwowzoru, stanowiąc *de facto* odrębny produkt. Próba uwzględnienia takich odstających wariantów (*outliers*) będzie stanowić duży problem przy odtwarzaniu uogólnionej architektury. W [28] proponuje się ich oddzielenie, aby finalna architektura SPL miała lepszą jakość. Na początku pozyskuje się informację strukturalną z każdego z wariantów. Uogólnia się też nazwy klas (manualnie), aby poprawnie identyfikować odpowiadające sobie klasy w różnych wariantach pomimo zmienionych nazw (w procesie powstawania wariantów). Informacje te są porównywane ze sobą pod kątem zmienności, filtrując niepasujące warianty za pomocą formalnej analizy konceptualnej (FCA). Decyzje o wyodrębnieniu komponentu (pakietu Javy) i o wykryciu bytów odstających zapadają na podstawie analizy progów określonych metryk. Wytworzona uogólniona architektura SPL ma postać: macierzy struktury (DSM – *design structure matrix*), widoku pakietów i klas z zaznaczonymi elementami obowiązkowymi i opcjonalnymi, siatki konceptów (*concept lattice*). Metoda jest nieodporna na podobne (a nie identyczne) odpowiadające sobie fragmenty kodu w różnych wariantach.

4.5. Analiza klonów

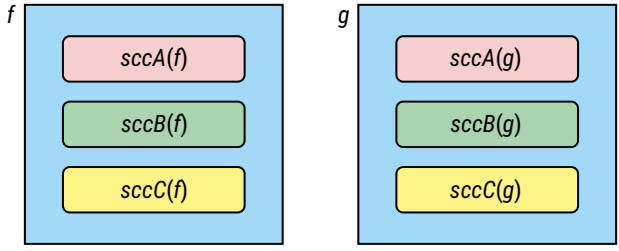
Podobieństwa w systemach informatycznych są powszechne, na różnych poziomach (wymagania, modele, kod źródłowy, …). W tej pracy rozważane będą klony programowe (*code clones*), które występują w kodzie źródłowym oprogramowania. Wg różnych oszacowań w dużych systemach sklonowany kod stanowi 20%–50% całości [29]. Ich źródłem najczęściej jest programowanie w stylu *copy-paste-modify*, choć nie można też wykluczyć podobnych fragmentów napisanych niezależnie. W niektórych przypadkach istnienie klonów jest zamierzone i pożądane [29], jednak w większości sytuacji dąży się do wyeliminowania istnienia klonów, ponieważ podobne struktury programowe są źródłem redundancji, co zwiększa objętość kodu, a co gorsza powoduje problemy w zarządzaniu nim (modyfikacje, naprawy błędów, niezależne refaktoryzacje, …). W tym celu w procesie refaktoryzacji można zaproponować generyczne, adaptowalne do różnych wymagań komponenty, których sparametryzowane użycie w miejsce klonów likwiduje redundancję. Aby jednak rozpocząć takie prace, należy te klony zidentyfikować. Zadanie to stanowi przedmiot licznych prac badawczych [30].

Identyfikacja identycznych fragmentów kodu jest często niewystarczająca. Oczywiście techniki detekcji klonów uwzględniają drobne zmiany w kodzie: usunięcia, wstawienia, zmiany kolejności fragmentów, zmiany nazw. Jednak w przypadku nieidentycznego kodu pojawia się problem, od kiedy uznać, że podobieństwo jest wystarczające, aby uznać, że fragmenty kodu stanowią klon. Proponowane metryki mogą określić wymagany próg podobieństwa, ale ich dobór zawsze jest subiektywny.

Zidentyfikowane, wzajemnie podobne do siebie miejsca w różnych fragmentach oprogramowania stanowią pojedynczą klasę klonów (*clone class*), a wyżej wymienione podobne miejsca są instancjami tej klasy.

4.5.1. Klony strukturalne

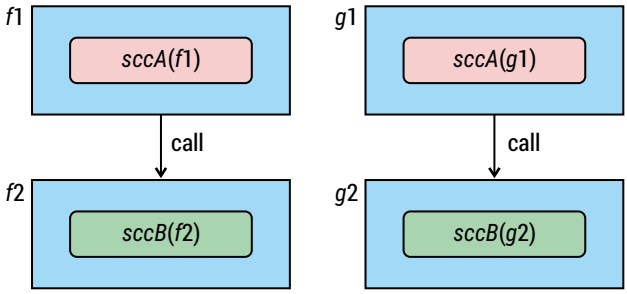
Oprócz prostych klonów zlokalizowanych w pojedynczych jednostkach programowych (metodach, czasem klasach), wyróżnia się klony strukturalne [31]. Pozwalają one wykorzystać fakt, że często obserwowane są powtarzające się takie same sekwencje wystąpienia klonów (dokładniej – instancji klas klonów prostych – *SCC – simple clone class*) w różnych miejscach w kodzie. Przykładowo, jeśli w jednej funkcji *f* występuje sekwencja instancji klas klonów prostych *sccA*(*f*), *sccB*(*f*), *sccC*(*f*) a w innej funkcji *g* taka sama sekwencja instancji klas klonów prostych *sccA*(*g*), *sccB*(*g*), *sccC*(*g*), to można mówić o klonie strukturalnym (rys. 4.1). Wyróżnić można klasy klonów strukturalnych na poziomie funkcji/metod (sekwencje klonów prostych występują w pojedynczych funkcjach/metodach), klas (programowania obiektowego), plików, katalogów, …



RYSUNEK 4.1. Przykład strukturalnej klasy klonów na poziomie funkcji

4.5.2. Współpracujące klony strukturalne

Jeśli dodatkowo sekwencje instancji klas klonów prostych występują w odpowiadających sobie konfiguracjach powiązanych strukturach programowych, to można mówić o współpracujących klonach strukturalnych (*collaborative structural clones*) [31]. Przykładowo, jeśli jako struktury programowe weźmiemy funkcje powiązane relacją wywołania, i jeśli znajdziemy dwie pary funkcji: *f1* woła *f2*, zaś *g1* woła *g2*, przy czym w *f1* i w *g1* zlokalizowane są 2 instancje jednej klasy klonów prostych *sccA*(*f1*) i *sccA*(*g1*) zaś w f2 i w g2 zlokalizowane są 2 instancje innej klasy klonów prostych *sccB*(*f2*) i *sccA*(*g2*), to taka konfiguracja stanowi klasę współpracujących klonów strukturalnych (rys. 4.2).



RYSUNEK 4.2. Przykład klasy współpracującego klonu strukturalnego występującego w parach wołających się funkcji

Odnajdywanie klonów strukturalnych, a zwłaszcza współpracujących klonów strukturalnych, może stanowić duże wsparcie do konstruowania generycznych komponentów do różnych zastosowań, np. linii produkcyjnych oprogramowania.

4.6. Analiza klonów w tworzeniu uogólnionej architektury

Zaproponowane ostatnio techniki odtwarzania uogólnione architektury wydają się działać poprawnie na przedstawionych przykładach [25, 27]. Nasuwa się jednak spostrzeżenie, że w zbyt małym stopniu wykorzystują one podobieństwa w kodzie analizowanych wariantów oprogramowania. Podobieństwo takie jest naturalną konsekwencją trybu powstawania tych wariantów: najczęściej *copy-paste-modify*. Rzadko kiedy kod pochodny jest modyfikowany w takim stopniu, żeby nie można było znaleźć podobieństw i skorzystać z nich. Podobieństwa takie mogą wskazać kandydatów dla generycznych modułów programowych na najniższym poziomie (szablony klas i metod/funkcji, klasy i metody generyczne, konstrukcje ART [17]). Pozwolić to może na tworzenie bibliotek używanych potem w wariantach, zmniejszając w nich redundancję i ogólnie podnosząc jakość. Zbuduje się też zasób generycznych modułów o dużym potencjale wielokrotnego użycia także w kolejnych produktach – skoro dowiodły swojej użyteczności w przeszłości. Jednak to tylko początek możliwości. Istniejące techniki w wielu miejscach odwołują się do analogicznych klas, metod występujących w wariantach. Jakże często podobieństwo to jest określane poprzez porównanie ich nazw lub identyczność kodu. Analiza klonów prostych może to zadanie wykonać o wiele lepiej, oferując porównanie zmodyfikowanych fragmentów kodu na poziomie pojedynczych modułów. Jeszcze większe możliwości oferują klony strukturalne, badające konfiguracje klonów prostych i wykazujące podobieństwo modułów na wyższym poziomie abstrakcji. Ale to współpracujące klony strukturalne, które analizują całe struktury modułów, wydają wprost stworzone do użycia w przypadku odkrywania natury wariantów. Ich analiza na poziomie wyższym niż pojedyncze klasy czy pliki wydaje się idealnie pasować do konstrukcji architektury linii

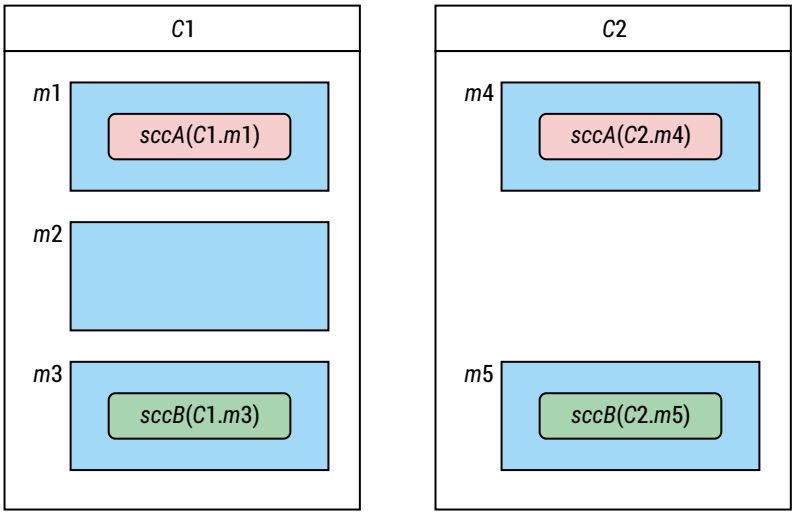
produkcyjnych oprogramowania, gdzie rozpatrywane są całe komponenty, zbudowane przecież ze współpracujących modułów programowych. Zwykle ich identyfikacja jest oddzielnym etapem proponowanych metod – tu gotowe uogólnione komponenty może zaproponować analiza klonów.

4.6.1. Użycie analizy klonów w istniejących technikach tworzenia uogólnionej architektury

Można tu wskazać kilka wymienionych wyżej technik tworzenia uogólnionej architektury, w których analiza klonów mogłaby poprawić działanie.

Technika oparta na uogólnionej metodzie refleksji Murphy'ego [24, 25] używa do znalezienia odpowiadających sobie modułów programowych (np. funkcji) w różnych wariantach produktu analizy klonów prostych. Obecność instancji klonów w porównywanych fragmentach kodu stwierdza się po przekroczeniu przez wybrane metryki (np. liczba linii kodu, instrukcji lub tokenów w zlokalizowanym klonie) określonych progów. Jeśli progi mają zbyt niską wartość, wykrytych klonów jest zbyt dużo, dlatego część z nich odrzuca się przez podwyższenie progu. Z drugiej strony zbyt wysoka wartość progu spowodować może przeoczenie ważnych par (lub zbiorów) podobnych modułów, które są swoimi odpowiednikami w wariantach produktów. Małe klony mogą występować w modułach seriami, przedzielone jednak wstawkami różniącego się kodu wystarczająco dużymi, aby przeszkodzić w uznaniu całości za instancję jednego prostego klonu. Klony strukturalne, skomponowane z sekwencji klonów prostych (próg akceptacji może być niski), stanowią swoisty filtr, odrzucający pojedyncze małe klony i przepuszczający ich sekwencje. Dobór progu akceptacji klonów prostych staje się mniej ważny – może być ustawiony na niższą wartość bez obawy o wskazanie zbyt wielu par (zbiorów) skojarzonych modułów programowych. Zastosowanie analizy klonów strukturalnych w miejsce klonów prostych ma potencjał poprawienia jakości wyprodukowanej uogólnionej architektury.

W [28] występuje manualny proces uogólnienia nazw klas, które podczas tworzenia nowych wariantów uległy zmianie. Klasy o takich samych nazwach w odrębnych wariantach uznawane są za odpowiadające sobie, więc ich ujednolicenie jest kluczowe w identyfikacji uogólnionych komponentów. Jak sami autorzy przyznają, podobne, choć różne fragmenty kodu mogą powodować błędne działanie metody. Remedium na to wydaje się proste – analiza klonów. Odpowiadające sobie klasy o różnych nazwach z łatwością zostaną zidentyfikowane przez analizę współpracujących klonów strukturalnych. Wystarczy jako struktury programowe (wskazane przykładowo na rys. 4.2 jako funkcje) przyjąć metody (i ewentualnie dodatkowo pola), a jako zależności pomiędzy nimi (wskazane na rys. 4.2 jako wywoływanie się) – przynależność do tej samej klasy – rys. 4.3. Byłoby to rozwiązanie automatyczne, w miejsce proponowanego manualnego ujednolicania nazw. Nazwy metod również nie miałyby tu znaczenia – liczyłoby się jedynie istnienie porównywalnych konfiguracji klonów.



RYSUNEK 4.3. Identyfikacja odpowiadających sobie klas *C1* i *C2* o różnych nazwach, metodach ale wykazujących istnienie współpracujących klonów strukturalnych. Takie współpracujące klony strukturalne mogą stanowić klasę klonów klasowych (CCC)

W [27] po odtworzeniu architektury wszystkich wariantów produktów identyfikowane są uogólnione komponenty przez hierarchiczne grupowanie (klastering) konkretnych komponentów z różnych wariantów produktów. Podobieństwo komponentów (potrzebne do wyboru scalanych w danym kroku – najbardziej podobnych) określa się poprzez kosinusową miarę podobieństwa określoną na nazwach klas-serwisów oferowanych przez każdy komponent.

Miara ta, używana w określaniu podobieństwa dokumentów tekstowych, jest wyrażona przez kosinus kąta pomiędzy dwoma wektorami zawierającymi częstotliwość występowania słów w obu dokumentach. Po przystosowaniu do komponentów i klas, dwa wektory reprezentujące komponenty zawierają tyle pozycji, ile jest unikalnych klas (nazw klas w opisywanej metodzie) w obu komponentach. Pozycje wektora zawierają wartości 0 (jeśli komponent nie zawiera klasy przyporządkowanej tej pozycji) i 1 (jeśli komponent zawiera taką klasę).

W wersji użytej w opisywanej metodzie miara będzie niedoskonała, jeśli nazwy klas się zmienią (nie ma tu nawet manualnego ujednolicania nazw znanego z [28]). Ponadto, ta sama nazwa może ukrywać klasy, które zmieniły się wzajemnie tak bardzo, że nie powinny być uważane za podobne.

Tak jak opisano wyżej, odpowiadające sobie klasy można identyfikować poprzez użycie współpracujących klonów strukturalnych (rys. 4.3). Następnie można kontynuować używanie miary kosinusowej opierając się na określeniu odpowiadających sobie klas nie poprzez zgodność ich nazw, ale przez występowaniu w nich tych samych instancji współpracujących klonów strukturalnych. W obu wektorach będą zatem pozycje reprezentujące klasy współpracujących klonów strukturalnych: z wartościami 0 jeśli komponent nie zawiera tego klonu, i 1 jeśli zawiera go.

Zamiast tego można też zaproponować konstrukcję uogólnionych komponentów bezpośrednio z kodów źródłowych komponentów, bez przechodzenia przez fazę odtwarzania ich indywidualnych architektur (jak opisano poniżej).

Dodatkowo w [27] zidentyfikowane uogólnione komponenty opisywane w zakresie zewnętrznej zmienności określonej poprzez interfejsy (wymagane i oferowane). Interfejsy w wariantach komponentów są uznawane za odpowiadające sobie na podstawie podobieństw kodu źródłowego implementujących je metod. Tu również analiza klonów prostych oraz klonów strukturalnych bardzo dobrze spełniłaby swoją rolę.

4.6.2. Współpracujące klony strukturalne w konstrukcji uogólnionych komponentów

Można się pokusić o użycie współpracujących klonów strukturalnych, przeniesionych z poziomu zbioru metod w klasach (rys. 4.3) na poziom współpracujących (połączonych) klas (wraz z ich metodami i polami), aby wspomóc bezpośrednią konstrukcję uogólnionych komponentów bezpośrednio z kodów źródłowych.

Mamy tu do czynienia z dwiema siłami określającymi formowanie uogólnionych komponentów. Po pierwsze, aby komponent był uogólniony, musi mieć swoje instancje (konkretne komponenty) w wariantach produktów, w identycznej lub zbliżonej formie. Po drugie, aby klasy utworzyły komponent, muszą razem wykazywać pewne charakterystyki. Można tu za [32] wymienić: (1) autonomiczność (możliwość użycia bez dodatkowych zależności), (2) zdolność komponentu do łączenia z innymi komponentami (*composability*) (3) specyficzność (*specificity*) rozumianą jako (minimalny) zbiór realizowanych funkcjonalności, współzależnych od siebie i trudnych do rozdzielenia. Gdyby wyrazić te pożądane cechy dobrego komponentu w formie wymaganej (symetrycznej) relacji między jego klasami (definiujących strukturę), można by połączyć te 2 siły właśnie w analizę współpracujących klonów strukturalnych. Tylko klasy pozostające w takiej relacji byłyby rozpatrywane jako miejsce wystąpienia instancji współpracującego klonu strukturalnego, a sam klon wyznaczyłby uogólniony komponent.

Przykładowo, wywoływanie metody jednej klasy przez drugą mogłoby definiować symetryczną relację między nimi. Wyrażałaby ona dążenie do formowania autonomicznych komponentów, ograniczając ich zależność od zewnętrznych modułów. Jednak prawdopodobnie skutkowałaby formowaniem zbyt dużych komponentów, w skrajnym przypadku jednego komponentu zbudowanego ze wszystkich klas systemu – przecież klasy komponentu są używane (wołane) z innych komponentów, a wyżej wymieniona relacja chciałaby umieścić 2 klasy: wołaną i wołającą w jednym komponencie. Innym przykładem potencjalnej relacji jest korzystanie przez obie klasy z tego samego zasobu. Taki rodzaj relacji reprezentowałby dążenie do formowania spójnych komponentów, w których klasy/metody mają te same zależności. Jednak formowane komponenty mogłyby być tu zbyt małe, ignorując zależności innego rodzaju (tak jak jak rozważane wcześniej wołanie jednej klasy z drugiej). Jednym słowem

określenie takiej symetrycznej relacji między klasami, która skutkowałaby łączeniem klas w dobrej jakości komponenty a jednocześnie stanowiłaby podstawę do użycia współdzielonych klonów strukturalnych, wydaje się bardzo trudne.

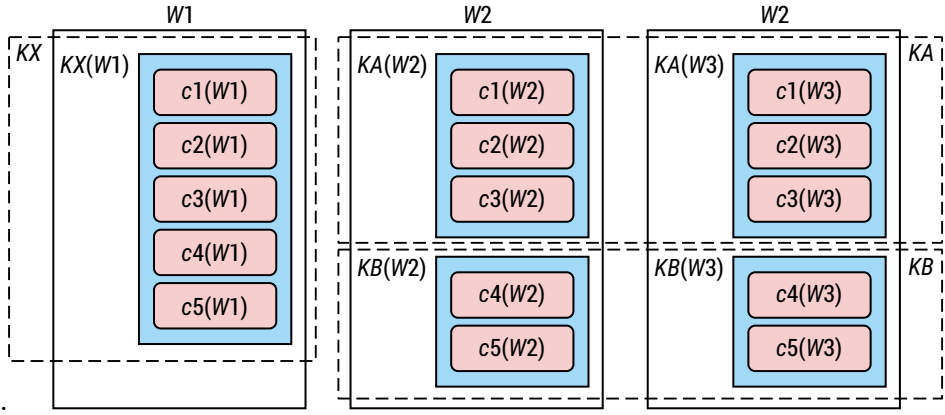
Jednak skoro określenie struktury za pomocą symetrycznej relacji jest tak trudne, to można by ją wyznaczyć w inny sposób – bezpośrednio za pomocą technik formujących komponenty w indywidualnych architekturach [32]. Klasy włączone do komponentów stanowiłyby struktury, w których szukałoby się współpracujących klonów strukturalnych pomiędzy wariantami. Znalezienie klasy takich klonów równałoby się wyodrębnieniu uogólnionego komponentu.

4.6.3. Bezpośrednie tworzenie uogólnionej architektury z uogólnionych komponentów

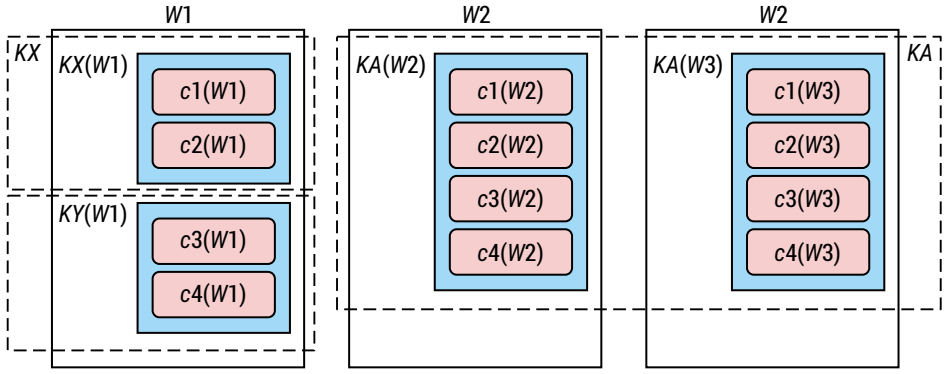
Opisane w podrozdziale 4.4 metody odtwarzania uogólnionej architektury wariantów oprogramowania zwykle zaczynają od odtworzenia indywidualnych architektur każdego wariantu. Dopiero z nich wydobywana jest informacja o elementach wspólnych i zmiennych. Na tym też etapie dochodzi do identyfikowania uogólnionych komponentów [27], gdzie konkretne komponenty wariantów są łączone w procesie hierarchicznego klasteringu. Podejście takie można określić mianem „**architektury najpierw**" (ARF – *architecture recovery first*). Ewentualne błędy popełnione na etapie formowania konkretnych komponentów skutkują błędnym wyodrębnieniem komponentów uogólnionych. Można sobie wyobrazić sytuację, gdzie konkretny komponent *KX*(*W1*) w jednym wariancie *W1* powstał przez połączenie zbyt wielu klas, które w innych wariantach *Wx* (x =/= 1) znalazły się w dwóch odrębnych komponentach *KA*(*Wx*) i *KB*(*Wx*). Taki zbyt duży komponent, albo zostanie zaliczony do uogólnionego komponentu *KA,* albo *KB*, albo też spowoduje powstanie uogólnionego komponentu *KX* (obecnego tylko w jednym wariancie *Wx*) – rys. 4.4. Taki „uogólniony" komponent w zasadzie nie zasługuje na to miano, gdyż jest obecny tylko w jednym wariancie. Jego potencjał powtórnego wykorzystania (*reusability*) jest ograniczony – może on znaleźć zastosowanie tam, gdzie powinny zostać użyte 2 niezależne komponenty *KA* i *KB*, konkurując z nimi. Jednak dwa niezależne komponenty będą tu lepsze niż jeden zespolony, gdyż bardziej uniwersalne, możliwe do zastosowania niezależnie od siebie, kiedy funkcjonalność oferowana przez drugi z pary jest niepotrzebna. Dlatego istnienie *KX* jest nadmiarowe.

Może też być tak, że w wariancie *W1* powstaną 2 odrębne, zbyt małe komponenty *KX*(W1) i *KY*(*W1*), które w innych wariantach *Wx* zostaną włączone do jednego komponentu *KA*(*Wx*). Przy formowaniu uogólnionych komponentów, do KA zostanie włączony albo *KX*(*W1*), albo *KY*(*W1*), albo żaden. Niewłączone komponenty z *W1* prawdopodobnie sformują niezależne uogólnione komponenty *KX* lub/i *KY* obecne tylko w *W1* – rys. 4.5. Można tu przez analogię z poprzednim przypadkiem stwierdzić, że 2 mniejsze komponenty będą lepsze niż jeden, zespolony. Jednak fakt, że w większości przypadków klasy *c1-c4* zostały włączone

do jednego komponentu pozwala sądzić, że jest to bardziej powszechna i ogólna sytuacja, zaś ich rozdzielenie w wariancie *W1* jest albo wynikiem błędu, albo bardzo szczególnej konfiguracji.



RYSUNEK 4.4. Zbyt duży komponent *KX*(*W1*) w wariancie *W1*. Budujące go klasy wchodzą w skład mniejszych komponentów (*KA*(*Wx*) i *KB*(*Wx*), x=2,3) w pozostałych wariantach

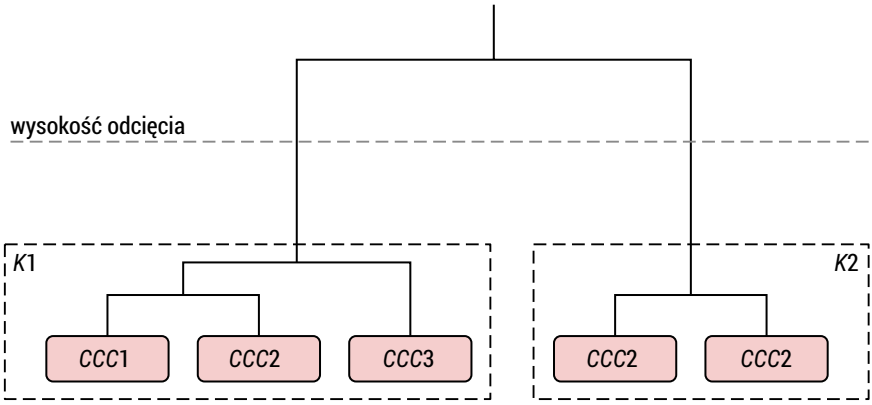


RYSUNEK 4.5. Zbyt małe komponenty *KX*(*W1*) i *KY*(*W1*) w wariancie *W1*. Budujące je klasy wchodzą w skład pojedynczego komponentu *KA*(*Wx*), x=2,3, w pozostałych wariantach

Łączenie komponentów między wariantami produktów na zasadzie: maksymalnie 1 komponent z każdego wariantu, nie pozwala na naprawienie błędów z etapu budowania niezależnych architektur wariantów produktów – podejmowane decyzje mogą być niespójne. Budowanie konkretnych komponentów dla każdego wariantu nie bierze pod uwagę ich późniejszego scalania – zamiast celu globalnego realizowany jest szereg celów lokalnych.

Zamiast tego można zaproponować bezpośrednią budowę komponentów poprzez hierarchiczne grupowanie klas klonów klasowych (CCC – *class clone class*). Takie klasy klonów klasowych powstawałyby jako współpracujące klony strukturalne na poziomie klas (rys. 4.3). Proces hierarchicznego klastrowania mógłby przebiegać

analogicznie jak w [32] i opierać się na maksymalizacji globalnej funkcji dopasowania oceniającej jakość podziału systemu (linii produkcyjnej oprogramowania) na komponenty (komponenty uogólnione). Wychodząc od pojedynczych CCC jako początkowych komponentów uogólnionych (liści w dendrogramie), w każdym kroku algorytmu łączone byłyby te liście lub węzły dendrogramu (rys. 4.6), dla których funkcja dopasowania osiągałaby maksymalną wartość. Wymagałoby to oczywiście sprawdzenia wszystkich potencjalnych złączeń. Wysokość odcięcia, decydująca o ostatecznym podziale na komponenty (powstałe w wyniku spłaszczenia gałęzi istniejących na wysokości odcięcia – rys. 4.6, jest ustalana w miejscu gdzie połączenie węzłów skutkuje gorszym komponentem wynikowym.



RYSUNEK 4.6. Formowanie uogólnionych komponentów (*K*1 i *K*2) z uogólnionych klas – klas klonów klasowych (CCC). Dendrogram – wynik hierarchicznego klastrowania – jest obcinany na wyliczonej wysokości odcięcia wyznaczając jednopoziomowe komponenty uogólnione

W cytowanej pracy globalna funkcja dopasowania liczy komponenty, których lokalna funkcja dopasowania przekracza wybrany próg. Lokalna funkcja dopasowania powinna osiągać wysokie wartości dla komponentów, które są autonomiczne, specyficzne (atomowe) i zdolne do komponowania z innymi (*composable*). Metryki używane do mierzenia tych charakterystyk są wyliczane na podstawie: liczby oferowanych przez komponent interfejsów, spoistości (LCC – *loose class cohesion*) i współzależności (*coupling*). Wyliczenie lokalnej funkcji dopasowania dla uogólnionych komponentów wymagałoby uogólnienia metryk, zaproponowanych oryginalnie dla indywidualnych komponentów. Uogólnienie to mogłoby przyjąć postać średniej, wartości maksymalnej lub minimalnej w każdym z wariantów.

Podejście takie, gdzie analiza klonów programowych wykonywana jest przed tworzeniem (tu – pojedynczej) architektury to może być określone mianem „**klony najpierw**" (CF – *clones first*). W wyniku bezpośredniej konstrukcji ujednoliconej architektury, bez przechodzenia przez architektury indywidualne, większa byłaby szansa na podjęcie optymalnej decyzji dotyczącej formowania uogólnionych komponentów, unikając problemów opisanych wyżej.

Podsumowanie

W pracy przedstawiona została analiza możliwości użycia analizy klonów w tworzeniu uogólnionej architektury wariantów oprogramowania na podstawie kodów źródłowych. Zaproponowane zostały (w podrozdziale 4.6) autorskie pomysły na zmodyfikowanie istniejących technik tworzenia takiej architektury poprzez uwzględnienie informacji o podobieństwach w kodzie pomiędzy wersjami programów. Architektura taka może być użyta przy konstruowaniu ujednoliconej linii produkcyjnej oprogramowania (SPL), ale też w innych zadaniach, jak remodularyzacja, wyodrębnianie serwisów (np. w celu przeniesienia oprogramowania do środowiska chmurowego).

Analiza klonów może być zastosowana do zmodyfikowania istniejących metod tworzenia uogólnionej architektury – wszędzie tam, gdzie zachodzi potrzeba odwołania się do odpowiadającego sobie kodu pomiędzy wariantami. Może też posłużyć do zaproponowania całkiem nowych mechanizmów, takich jak bezpośrednie formowanie uogólnionych komponentów z uogólnionych klas, zidentyfikowanych przy użyciu koncepcji współpracujących klonów strukturalnych – bez pośredniego etapu budowy indywidualnych architektur wariantów.

Propozycje zostały tu przedstawione w formie koncepcji i wymagają doprecyzowania i eksperymentalnej weryfikacji. Jednak ich opracowanie oparte zostało na istniejących i opublikowanych rozwiązaniach w zakresie analizy klonów i tworzenia uogólnionej literatury, co pozwala mieć nadzieję na pozytywne rezultaty. Obecnie trwają prace na zaimplementowaniem wybranych technik w celu oceny korzyści z użycia informacji o klonach w tym zadaniu (głównie w oparciu o modyfikację algorytmu zaprezentowanego w [27]).

Badania zostały zrealizowane w ramach pracy nr WZ/WI-IIT/3/2020, sfinansowanej ze środków Ministerstwa Nauki i Szkolnictwa Wyższego w Polsce.

Bibliografia

[1] Garlan D., Shaw M., *An Introduction to Software Architecture*. School of Computer Science, Carnegie Mellon University, 1994

[2] *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*, IEEE Std, 2000, 1471–2000

[3] Kruchten P., *The Rational Unified Process: An Introduction*, Addison-Wesley Professional, 2004

[4] Kruchten P., *Architectural Blueprints – The "4+1" View Model of Software Architecture*, IEEE Software, 1995, 12 (6), pp. 42–5

[5] Fowler M., *Who Needs an Architect?* IEEE Software, 2003, 20(5), pp. 11–13

[6] Koschke R., *Architecture Reconstruction. Software Engineering*, International Summer Schools, ISSSE 2006–2008, Revised Tutorial Lectures, 2009, pp. 140–173

[7] Garcia J., Ivkovic I., Medvidovic N., *A comparative analysis of software architecture recovery techniques*, 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013, pp. 486–496

[8] Maqbool O., Babri H.A., *Hierarchical Clustering for Software Architecture Recovery*. IEEE Transactions On Software Engineering, 2007, 33(11), pp. 759–780

[9] Praditwong K., *Solving software module clustering problem by evolutionary algorithms*, 8th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2011, pp. 154–159

[10] Sartipi K., Kontogiannis K., *A graph pattern matching approach to software architecture recovery*, IEEE International Conference on Software Maintenance, 2001, pp. 408–419

[11] Mitchell B.S., Mancoridis S., *On the Automatic Modularization of Software Systems Using the Bunch Tool*, IEEE Transactions On Software Engineering, 2006, 32(3), pp. 193–208

[12] Xiao C., Tzerpos V., *Software clustering based on dynamic dependencies*, Ninth European Conference on Software Maintenance and Reengineering, 2005, pp. 124–133

[13] Anquetil N., Lethbridge T., *Recovering Software Architecture from the Names of Source Files*, Journal of Software Maintenance Research and Practice, 1999, 11(3), pp. 201–221

[14] Clements P., Northrop L., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002

[15] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994

[16] Zhang H., Jarzabek S., *XVCL: a mechanism for handling variants in software product lines*, Science of Computer Programming, 2004, 53(3), pp. 381–407

[17] Adaptive Reuse Technology, http://art-processor.org

[18] Weiss D.M., Lai C.T.R., *Software Product-line Engineering*: *A Family-based Software Development Process*. Addison-Wesley, 1999

[19] Krueger C., *Easing the Transition to Software Mass Customization*, [in:] van der Linden F. (eds) Software Product-Family Engineering. PFE 2001. Lecture Notes in Computer Science 2290. Springer, 2002

[20] Bayer J., Flege O., Knauber P., Laqua R., Muthig D., Schmid K., Widen T., DeBaud J.M., *PuLSE: methodology to develop software product lines*, Symposium on Software reusability, 1999, pp. 122–131

[21] Stoermer C., O'Brien L., *MAP – mining architectures for product line evaluations*, Working IEEE/IFIP Conference on Software Architecture, 2001, pp. 35–44

[22] Pinzger M., Gall H., Girard J.-F., Knodel J., Riva C., Pasman W., Broerse C., Wijnstra J.G., *Architecture recovery for product families*, Software product-family engineering. Springer, 2004, pp. 332–351

[23] Kang K.C., Kim M., Lee J., Kim B., *Feature-oriented re-engineering of legacy systems into product line assets–a case study*, International Conference on Software Product Lines (SPLC2005). Springer, 2005, pp. 45–56

[24] Frenzel P., Koschke R., Breu A.P.J., Angstmann K., *Extending the Reflexion Method for Consolidating Software Variants into Product Lines*, 14th Working Conference on Reverse Engineering (WCRE 2007), 2007, pp. 160–169

[25] Koschke R., Frenzel P., Breu A.P., Angstmann K., *Extending the reflexion method for consolidating software variants into product lines*, Software Quality Journal, 2009, 17 (4), pp. 331– 366

[26] Linsbauer L., Lopez-Herrejon R.E., Egyed A., *Variability extraction and modeling for product variants*, Software & Systems Modeling 16, 2017, pp. 1179–1199

[27] Shatnawi A., Seriai A.-D., Sahraoui H., *Recovering software product line architecture of a family of object-oriented product variants*, Journal of Systems and Software 131, 2017, pp. 325–346

[28] Lima C., Assunção W.K., Martinez J., Mendonça W., Machado I.C., Chavez C.F.G., *Product line architecture recovery with outlier filtering in software families: the Apo-Games case study*, Journal of the Brazilian Computer Society 25, article 7, 2019

[29] Jarzabek S., *Software Similarities and Clones: A Curse or Blessing?* International Conference on Enterprise Information Systems (ICEIS 2020), Keynote, 2020

[30] Hammad M., Basit H., Jarzabek S., Koschke R., *A Systematic Mapping Study of Clone Visualization*, Computer Science Review 37, 2020, pp. 1–55

[31] Basit H.A., Jarzabek S., *Data Mining Approach for Detecting Higher-level Clones in Software*, IEEE Transactions On Software Engineering, 2009, 35(4), pp. 497–514

[32] Kebir S., Seriai A.-D., Chardigny S., Chaoui A., *Quality-centric approach for software component identification from object-oriented code*, Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA), 2012, pp. 181–190

[33] Faust D., Verhoef C., *Software product line migration and deployment*, Journal of Software Practice and Experiences 33(10), 2003, pp. 933–955

[34] Mancoridis S., Mitchell B., Chen Y.-F., Gansner E., *Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures*, IEEE International Conference on Software Maintenance (ICSM'99), 1999, pp. 50–59

Spis tabel

Tabela 1.1. Średnie czasy iteracji (w sekundach) dla różnych algorytmów k-średnich w zależności od liczby klastrów K oraz liczby wektorów uczących N (zbiory mix10) 19

Tabela 1.2. Średnie czasy iteracji (w sekundach) dla różnych algorytmów k-średnich w zależności od liczby klastrów K oraz liczby wektorów uczących N (zbiory mix300) 22

Tabela 2.1. Porównanie skuteczności systemów biometrycznych bazujących na bazie [4] 32

Tabela 2.2. Architektura *Systemu I* oraz *Systemu II*, *in/out – wejściowa/wyjściowa liczba filtrów, ks- kernel size, p-padding, s-stride* 41

Tabela 2.3. Architektura *Systemu III* 42

Tabela 2.4. Zestawienie skuteczności opracowanych systemów biometrycznych 43

Tabela 3.1. Wynik dla zbioru Z1 względem parametrów Δr i σG 60

Tabela 3.2. Wynik dla zbioru Z2 względem parametrów Δr i σG 61

Tabela 3.3. Wynik dla zbioru Z3 – parametry Δr i σG 61

Tabela 3.4. Wyniki testu – filtr antyrefleksu i ograniczenie obszaru poszukiwań, zbiór Z1 62

Tabela 3.5. Wyniki testu – filtr antyrefleksu i ograniczenie obszaru poszukiwań, zbiór Z2 63

Tabela 3.6. Wyniki testu – filtr antyrefleksu i ograniczenie obszaru poszukiwań, zbiór Z3 63

Tabela 3.7. Wyniki testu zbiór Z1 – adaptacyjne wyrównywanie histogramu 64

Tabela 3.8. Wyniki testu zbiór Z2 – adaptacyjne wyrównywanie histogramu 65

Tabela 3.9. Wyniki testu zbiór Z3 – adaptacyjne wyrównywanie histogramu 65

Spis rysunków

Rysunek 1.1. Architektura procesora AMD Ryzen Threadripper 1950X [17] 16
Rysunek 1.2. Schemat budowy bloku SM w architekturze Turing [18] 17
Rysunek 1.3. Przyspieszenie trzech algorytmów k-średnich
nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 1,5 mln 20
Rysunek 1.4. Przyspieszenie trzech algorytmów k-średnich
nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 15 mln 21
Rysunek 1.5. Przyspieszenie trzech algorytmów k-średnich
nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 150 mln 22
Rysunek 1.6. Przyspieszenie trzech algorytmów k-średnich
nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 50 tys 23
Rysunek 1.7. Przyspieszenie trzech algorytmów k-średnich
nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 500 tys 24
Rysunek 1.8. Przyspieszenie trzech algorytmów k-średnich
nad algorytmem Lloyda (CPU) dla różnych wartości K oraz N = 5 mln 25
Rysunek 2.1. Schemat blokowy systemu biometrycznego..... 34
Rysunek 2.2. Technika uzupełniania zerami o wymiarze
a) 3×N, b) 2×N, oraz filtach splotowych o wymiarach 4×4 37
Rysunek 2.3. Sygnały zarejestrowane podczas krótkiej przerwy w chodzie:
a) sygnał magnitudy akcelerometru; za pomocą szarego tła zaznaczono okresy
uznawane za aktywny chód b) sygnały zarejestrowane przez poszczególne osie czujnika..... 38
Rysunek 2.4. Długość nagrań zgromadzonych
dla poszczególnych uczestników badań przed oraz po zastosowaniu segmentacji 39
Rysunek 2.5. Struktura danych wejściowych *Systemu I*..... 40
Rysunek 2.6. Struktura danych wejściowych *Systemu II* 41
Rysunek 2.7. Przebieg łączenia danych wejściowych dla *Systemu III*..... 42
Rysunek 3.1. Obraz obu oczu z zaznaczonymi charakterystycznymi miejscami 49
Rysunek 3.2. Schemat blokowy typowego systemu rozpoznawania tęczówki 50
Rysunek 3.3. Tęczówka oka w bliskiej podczerwieni i świetle widzialnym [7] 52
Rysunek 3.4 Krawędzie na obrazie tęczówki
uzyskane z użyciem operatora Sobel'a w osi x i y 53

Rysunek 4.1. Przykład strukturalnej klasy klonów na poziomie funkcji 75

Rysunek 4.2. Przykład klasy współpracującego klonu strukturalnego występującego w parach wołających się funkcji 76

Rysunek 4.3. Identyfikacja odpowiadających sobie klas *C1* i *C2* o różnych nazwach, metodach ale wykazujących istnienie współpracujących klonów strukturalnych. Takie współpracujące klony strukturalne mogą stanowić klasę klonów klasowych (CCC) 78

Rysunek 4.4. Zbyt duży komponent *KX*(*W1*) w wariancie *W1*. Budujące go klasy wchodzą w skład mniejszych komponentów (*KA*(*Wx*) i *KB*(*Wx*), x=2,3) w pozostałych wariantach 81

Rysunek 4.5. Zbyt małe komponenty *KX*(*W1*) i *KY*(*W1*) w wariancie *W1*. Budujące je klasy wchodzą w skład pojedynczego komponentu *KA*(*Wx*), x=2,3, w pozostałych wariantach 81

Rysunek 4.6. Formowanie uogólnionych komponentów (*K*1 i *K*2) z uogólnionych klas – klas klonów klasowych (CCC). Dendrogram – wynik hierarchicznego klastrowania – jest obcinany na wyliczonej wysokości odcięcia wyznaczając jednopoziomowe komponenty uogólnione 82

Politechnika Białostocka