Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Praca magisterska

Optymalizacja algorytmu wyznaczającego przepływ optyczny dla obrazów kolorowych

Szczepan Kurnyta

Nr albumu: 120867
Kierunek: Elektrotechnika
Specjalność: Inżynieria komputerowa w przemyśle

Promotor: dr inż. Jarosław Bułat

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Kraków 2008

Oświadczenie autora

Ja, niżej podpisany Grzegorz Suder oświadczam, że praca ta została napisana samodzielnie i wykorzystywała (poza zdobytą na studiach wiedzą) jedynie wyniki prac zamieszczonych w spisie literatury.

. . . . . .
(Podpis autora)

Oświadczenie promotora

Oświadczam, że praca spełnia wymogi stawiane pracom magisterskim.

. . . . . .
(Podpis promotora)

Spis treści

Wstęp

Poniższa praca opisuje problematykę obliczania i metod optymalizacji algorytmów wyznaczających przepływ optyczny.

Cyfrowa analiza i przetwarzanie obrazów w dzisiejszych czasach stosowana jest w medycynie, przemyśle i nauce. Coraz większa dostępność i niska cena systemów wizyjnych powodują, że chętnie są one używane w wielu aplikacjach wspomagających pomiary, sterowanie czy kontrolę, a także okazują się być bardzo dobrym narzędziem służącym automatyzacji procesów przemysłowych. Ciągle udoskonalane i rozwijane są zastosowania w różnych rodzajach tomografii (np. PET, ang. Positron Emission Tomography) i nawigacji operacyjnej, podczas wykonywania skomplikowanych zabiegów. Wspomaganie lekarza poprzez symulację i planowanie operacji, a także wizualizacja, śledzenie i automatyczna korekcja paramtrów aparatury to elementy, które niesłychanie ułatwiają przeprowadzenie zabiegu. Jednymi z wykonywanych badań z tej tematyki są prace nad obliczaniem i optymalizacją wyznaczania przepływu optycznego. Jego estymacja jest wykorzystywana między innymi do rozpoznawaniu obiektów, wykrywaniu ruchu, kompresji video, estymacji obiektów 3D z sekwencji obrazów 2D. Zagadnienie wyznaczania przepływu optycznego jest intensywnie badane od początku lat 80-tych i ciągle rozwijane do wielu zastosowań.

Estymacja ruchu opisywana w tej pracy ma służyć do pozycjonowania kamery bronchofiberoskopu w drzewie oskrzelowym podczas zabiegu bronchoskopii aspiracyjnej. Dokładność obliczania przepływu optycznego metodami siłowymi typu „brute force” pozwala otrzymać pożądany efekt pod względem dokładności, jednak jest to okupione długim czasem obliczeń. Konieczne jest zaproponowanie takich sposobów, które z równą precyzją, ale w czasie rzeczywistym pozwolą nam wyznaczać wektory ruchu. Znaczne przyspieszenie obliczeń można osiągnąć poprzez zastosowanie metod minimalizacji wieloparametrowej jak np. algorytmy gradientowe i bezgradientowe testowane w tej pracy. Innym podejściem jest użycie wyszukiwania połączonego z dekompozycją falkową badanych obrazów. Dodatkowym wsparciem może być użycie specjalnych bibliotek numerycznych, które wykorzystają optymalnie architekturę komputera, skracając czas wykonywania programu.

Prezentowana praca składa się z części teoretycznej i części praktycznej. Pierwsze dwa rozdziały przedstawiają problematykę zagadnienia oraz proponowane metody jego rozwiązania. Jest w nich zawarty opis poszukiwanego przepływu optycznego, sposób badania i charakterystyczne cechy. Pierwszy rozdział jest wzbogacony o informacje dotyczące dekompozycji obrazów i ich interpolacji. W rozdziale drugim znajduje się przegląd zastosowanych w części autorskiej metod wraz z opisem znaczących różnic pomiędzy stosowanymi podejściami.

Praktyczna część zawiera programową realizację algorytmu wyszukiwania przepływu optycznego w języku C++, w środowisku kDevelop. Zaimplementowane zostały omówione metody tj.: wyszukiwanie siłowe, algorytm najszybszego spadku, metoda sympleksów Neldera-Meada. Ponadto programy zawierają procedury interpolacji, dekompozycji obrazów, a także funkcje wykorzystujące optymalne biblioteki GSL (GNU Scientific Library) i BLAS (Basic Linear Algebra Subprograms). Uzupełnieniem są informacje na temat możliwości zastosowania wielowątkowości, a także skrypty napisane w języku Matlab, które służyły do generowania materiału referencyjnego i pomiaru jakości podczas licznych testów.

Ostatni rozdział przedstawia wyniki przeprowadzonych badań i porównuje zaimplementowane algorytmy pod względem czasu wykonywania i dokładności. Dodatkowo zaprezentowane zostały różne warianty testowanych algorytmów. Ostatecznie przebadano wyznaczanie przepływu optycznego dla obrazów kolorowych oraz wpływ modeli barwnych na jakość estymacji. Założonym wynikiem tej pracy ma być wybór najlepszej z analizowanych metod zarówno pod względem szybkości jak i precyzyjności wyznaczania wektorów ruchu.

1 Wykorzystanie przepływu optycznego w algorytmach estymacji ruchu

W pierwszym rozdziale przedstawione zostaną podstawowe informacje dotyczące wyznaczania pola wektorów ruchu. W kolejnych podrozdziałach zaprezentowany zostanie opis przepływu optycznego i jego zastosowania. Następnie ukazane zostaną informacje dotyczące rodzaju i zasady działania, niezbędnej w programie, funkcji interpolującej. Kolejny podrozdział poświęcono na przedstawienie falkowej dekompozycji obrazów, jako podejście, które przyspiesza wykonywanie algorytmów obliczających przepływ optyczny i czyni je mniej wrażliwymi na znajdowanie minimów lokalnych zamiast globalnych. Uzupełnieniem są informacje na temat kryteriów porównywania bloków, sposobu badania przepływu optycznego i doboru obrazów testowych.

1.1 Przepływ optyczny

Poruszanie się obiektów jest nieodłączną częścią postrzeganego przez nas świata. Jest ono szczególnie ważnym źródłem informacji wykorzystywanych w wielu operacjach przetwarzania obrazów, takich jak: rozpoznawanie kształtów, śledzenie obiektów czy generowanie trójwymiarowych struktur na podstawie sekwencji zdjęć 2D [4]. U podstawy większości takich zastosowań leży obliczanie przepływu optycznego (ang. *Optical Flow*). To właśnie z badań nad nim rozwinęły się dziedziny estymacji ruchu czy kompresji video, dziś znajdujące szerokie, praktyczne zastosowanie.

Przepływ optyczny to inaczej pole wektorów ruchu wyznaczone pomiędzy dwoma obrazami w sekwencji video. Umożliwia ono przekształcenie pierwszego obrazu w drugi poprzez przemieszczenie między nimi odpowiednich obszarów zgodnie z wektorami pola. Początek i koniec każdego z wektorów odpowiadają jednemu przesuniętemu punktowi. Ważne jest wyznaczenie kierunku, zwrotu i długości każdego wektora, gdyż parametry te zawierają informację, w którą stronę przemieszcza się dany piksel i z jaką prędkością. Wektor prędkości dla przypadku 2D ma postać:

$$c = \left(\frac{dx}{dt}, \frac{dy}{dt} \right) \tag{1}$$

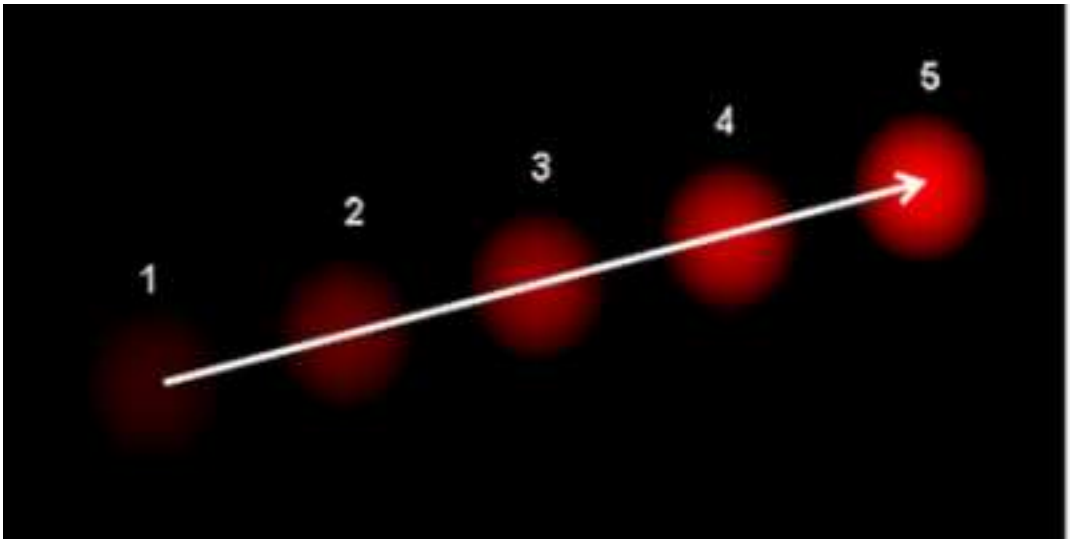
gdzie:

dx – przesunięcie piksela w kierunku OX,

dy – przesunięcie piksela w kierunku OY,

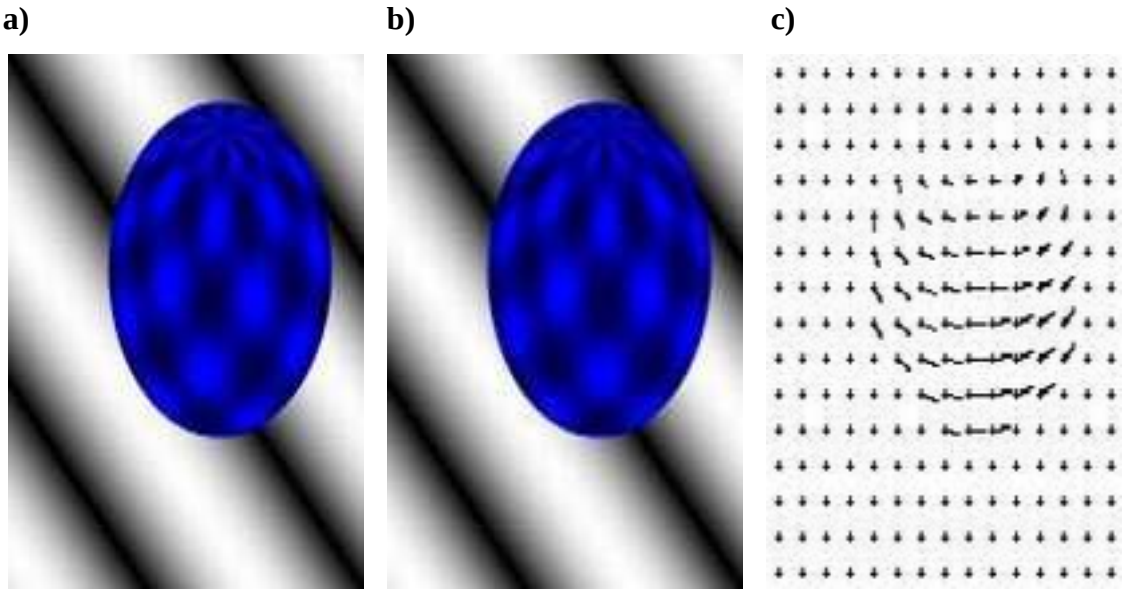
dt – czas pomiędzy obrazami w sekwencji.

Na rysunku 1.1 zaprezentowany został przykładowy wektor i obiekt, którego zmianę położenia ten wektor estymuje.



Rys. 1.1 Wektor ruchu obrazujący drogę czerwonego obiektu w sekwencji video [1].

Obliczenie przepływu optycznego dla dwóch obrazów polega na znalezieniu wielu takich wektorów, które ukazują rozkład pola na całym obszarze testowanych klatek video. W zależności od zastosowania liczba wektorów może być różna. Dobry przykład przepływu optycznego przedstawiono na rysunku 1.2.



Rys. 1.2 Wektory ruchu powstałe wskutek obrotu sfery: a) położenie pierwsze, b) położenie drugie, c) pole wektorów ruchu powstałe wskutek przemieszczenia [2].

1.1.1 Warunek przepływu optycznego

Przepływ optyczny jest definiowany jako prędkość ruchu wzorków (deseni) o zbliżonej jasności I w sekwencji obrazów [3]. W gradientowym podejściu do obliczania przepływu optycznego zakłada się, że jasność obrazu określa funkcja $I(x, y, t)$. Po upływie krótkiego czasu charakterystycznego dla wystąpienia kolejnej klatki w sekwencji video, w pobliżu punktu (x, y), jasność można przedstawić za pomocą wzoru:

$$I(x + dx, y + dy, t + dt) = I(x, y, t) + \frac{\partial I}{\partial x} dx + \frac{\partial I}{\partial y} dy + \frac{\partial I}{\partial t} dt \tag{2}$$

gdzie:

$\frac{\partial I}{\partial t}$ — szybkość zmiany jasności piksela w czasie,

$\frac{\partial I}{\partial x}$ — składowa zmiany jasności przy przesunięciu wzdłuż osi X,

$\frac{\partial I}{\partial y}$ — składowa zmiany jasności przy przesunięciu wzdłuż osi Y.

Przyjmując, że pewien fragment obrazu znajdujący się w okolicy punktu (x, y) w czasie t, przesunął się po upływie czasu dt o wektor (dx, dy) nie zmieniając swojej jasności, można napisać:

$$I(x + dx, y + dy, t + dt) = I(x, y, t) \tag{3}$$

Wtedy na podstawie (2) i (3) otrzymujemy:

$$\frac{\partial I}{\partial x} dx + \frac{\partial I}{\partial y} dy + \frac{\partial I}{\partial t} dt = 0 \tag{4}$$

z czego dzieląc przez dt:

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} \tag{5}$$

co można inaczej zapisać:

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v \tag{6}$$

gdzie:

$u = \frac{dx}{dt}$ — składowa prędkości piksela równoległa do osi X,

$v = \frac{dy}{dt}$ — składowa prędkości piksela równoległa do osi Y.

Otrzymane równanie (6) to warunek przepływu optycznego. Jest to najprostsza forma opisu matematycznego tego zjawiska.

1.1.2 Metody

Istnieją trzy główne grupy metod obliczania przepływu optycznego [3][4]:

- korelacyjne – bazujące na wzajemnej odpowiedniości (korelacji) bloków pikseli,
- częstotliwościowe – wykorzystujące szybki algorytm DFT do detekcji przesunięcia przestrzennego,
- gradientowe – opierające się na pochodnych przestrzennych intensywności obrazu.

Bardzo szeroko stosowane są obecnie metody z pierwszej grupy, wykorzystujące porównywanie bloków (ang. *Block Matching Algorithm*). Obrazy są wtedy dzielone na grupy MxM pikseli i dla każdego bloku z jednego obrazu jest poszukiwany najbardziej podobny blok na obrazie drugim. Metoda ta jest bardzo przydatna szczególnie kiedy obliczenie pochodnych (stosowanych przy metodach gradientowych), nie jest łatwe. Takie sytuacje mogą zdarzyć się często np. na skutek zbyt szybkiego ruchu lub braku wystarczającej ilości klatek w sekwencji. Sposób ten pozwala uzyskać również większą dokładność i jest dobrą strategią walki z szumem występującym w obrazach poprzez stosowanie bloków o większych rozmiarach. Niewątpliwą zaletą metod korelacyjnych jest ponadto możliwość wyboru ilości i dowolnego rozmieszczenia wektorów ruchu. Algorytmy te dążą do znalezienia maksimum funkcji korelacyjnej. Można ją przedstawić w następujący sposób [3]:

$$K(\vec{z}) = \int_R I_1(\vec{x}) I_2(\vec{x} + \vec{z}) d\vec{x} \tag{7}$$

gdzie:

$I_1(\vec{x})$ i $I_2(\vec{x})$ – funkcje intensywności pierwszego i drugiego obrazu,

$\vec{x} = (x_{x,} x_y)$ – wektor położenia,

$\vec{z} = (z_{x,} z_y)$ – wektor przemieszczenia,

R – przestrzeń obrazu.

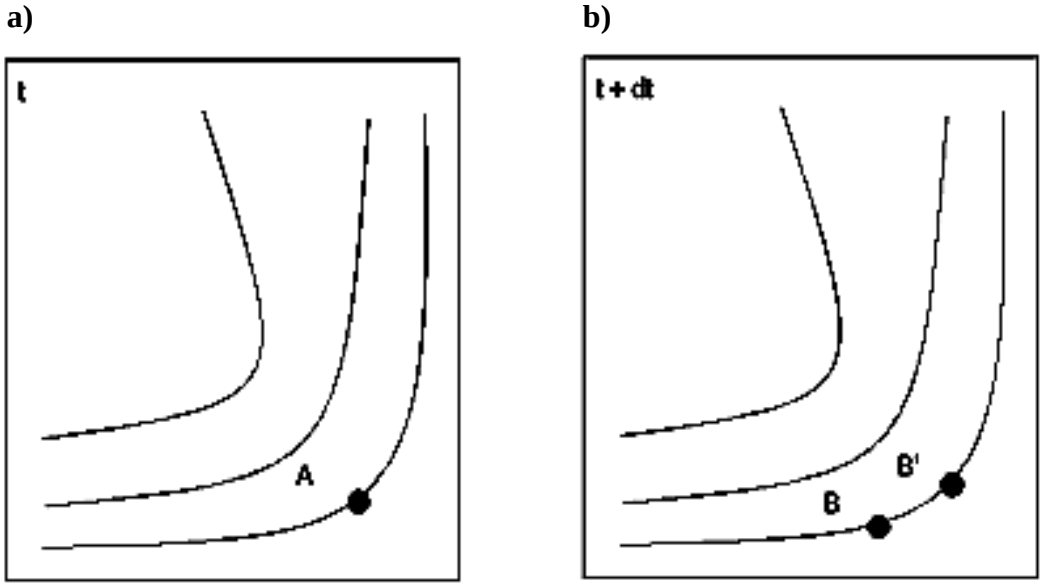
Założenia i cechy charakterystyczne metody korelacyjnej [3]:

- sąsiednie punkty poruszają się w tym samym kierunku i z równą prędkością,

— zwiększenie obszaru poszukiwań zapewnia znajdowanie dalszych przesunięć,
— zwiększenie wielkości porównywanych łatek, powoduje zmniejszenie występującego szumu, ale również wzrost możliwości wystąpienia błędów skorelowania.

Omawiana metoda znalazła m.in. zastosowanie do kompresji strumienia video w formacie MPEG. Takie podejście (porównywania bloków a nie poszczególnych pikseli) stosowane jest również w tej pracy we wszystkich używanych metodach wyszukiwania.

Inną grupę stanowią metody gradientowe, które zajmują się analizą pochodnych intensywności obrazu. W prezentowanym powyżej warunku przepływu optycznego występują pochodne czasowe i przestrzenne, a ich związek przedstawia (6). Wynika z niego, że często może występować niejednoznaczność nowego położenia poszukiwanego punktu. Opisywana sytuacja jest przedstawiona na rysunku 1.3. Ciągłe linie oznaczają kontury o tej samej intensywności, dlatego na podstawie (6) nie można dokładnie przewidzieć nowego położenia (ang. *Aperture Problem*), ponieważ punktowi A (rys. 1.3a) może odpowiadać punkt B lub B’ (rys. 1.3b).



Rys. 1.3 Linie oznaczające kontury o jednakowej intensywności: a) położenie punktu w czasie t – A, b) nowe prognozowane położenie punkty w czasie t+dt – B i B’.

Do weryfikacji obliczonego przepływu optycznego w omawianej metodzie służy odpowiednie wyrażenie nazywane również warunkiem gładkości [3]. Określa ono jak obliczone pole różni się od idealnego:

$$S = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \tag{8}$$

gdzie:

$u = \frac{\partial x}{\partial t}$ – składowa prędkości wektora równoległa do osi X,

$v = \frac{\partial y}{\partial t}$ – składowa prędkości wektora równoległa do osi Y.

Naturalnie pożądane jest, aby funkcja S przyjmowała jak najmniejszą wartość. Dodatkowo poszukiwane jest minimum funkcji C, określającej opisany w poprzednim paragrafie warunek przepływu (6):

$$C = \left(\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right)^2 \tag{9}$$

Z powodu konieczności uwzględnienia obu powyższych warunków, stosuje się metodę mnożników Lagrange'a, dzięki której minimalizowaną funkcję można przedstawić następująco:

$$F(u,v) = S(u,v) + \lambda C(u,v) \tag{10}$$

W (11) λ jest parametrem modyfikowanym ze względu na jakość obrazów. Im większy szum tym wartość parametru λ powinna być mniejsza [3]. Ostatecznie wstawiając (8) i (9) do (10) otrzymujemy:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= \lambda\left(\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right)\frac{\partial I}{\partial x} \\ \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} &= \lambda\left(\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right)\frac{\partial I}{\partial y} \end{aligned} \tag{11}$$

Zaletą tego układu jest możliwość iteracyjnego rozwiązania. Przedstawiony sposób nie jest wykorzystywany w tej pracy. Wprawdzie autor wykorzystuje, później opisywaną metodę gradientową i bezgradientową, ale są to metody minimalizacji zadanej funkcji celu i nie korzystają wprost z przedstawionego schematu obliczeń.

Trzecią grupę stanowią metody częstotliwościowe. Są to najbardziej skomplikowane sposoby estymacji przepływu optycznego, wykorzystywane przez

zaawansowane aplikacje. Mają m.in. zdolność do detekcji ruchu w scenach, w których zmienia się oświetlenie czy wykrywania obiektów poruszających się na tle różnobarwnych wzorów. Analizują one obraz w dziedzinie częstotliwości i wykorzystują filtry czułe na kierunek ruchu. Jedną z bardziej znanych metod w tej grupie jest wzajemna korelacja fazowa (ang. *Cross Power Spectrum*), stosująca szybkie algorytmy 2D FFT do wykrywania przesunięcia przestrzennego w różnicy widm fazowych fragmentów obrazów. Według wielu badaczy metoda częstotliwościowa jest najbliższa naturalnemu sposobowi detekcji ruchu, jakim posługuje się człowiek w procesie postrzegania. Wadą tej grupy metod jest duża złożoność obliczeniowa, która uniemożliwia analizę w czasie rzeczywistym. Z tego również względu omawiany sposób nie jest poruszany w praktycznej części tej pracy.

1.1.3 Siatka

Metody obliczania przepływu optycznego różnią się również w swobodzie doboru siatki punktów na podstawie których obliczane jest pole wektorów ruchu. W zależności od potrzeby zastosowania w podejściu gradientowym i korelacyjnym można skorzystać z siatki regularnej lub nieregularnej. Pierwsza z nich określa równomierne rozłożenie punktów na płaszczyźnie, natomiast druga umożliwia zagęszczenie, rozrzedzenie wyznaczanego przepływu w pożądanych miejscach. Siatka nieregularna możliwa jest do wyznaczenia ze względu na to, że metody korelacyjne i większość gradientowych, wyznacza każdy wektor ruchu osobno, bez związku z pozostałymi. Metody częstotliwościowe, wymuszają użycie wyłącznie regularnych siatek, ponieważ zazwyczaj korzystają z analizy Fouriera lub podobnej transformacji, która wymusza równomierne próbkowanie sygnału.

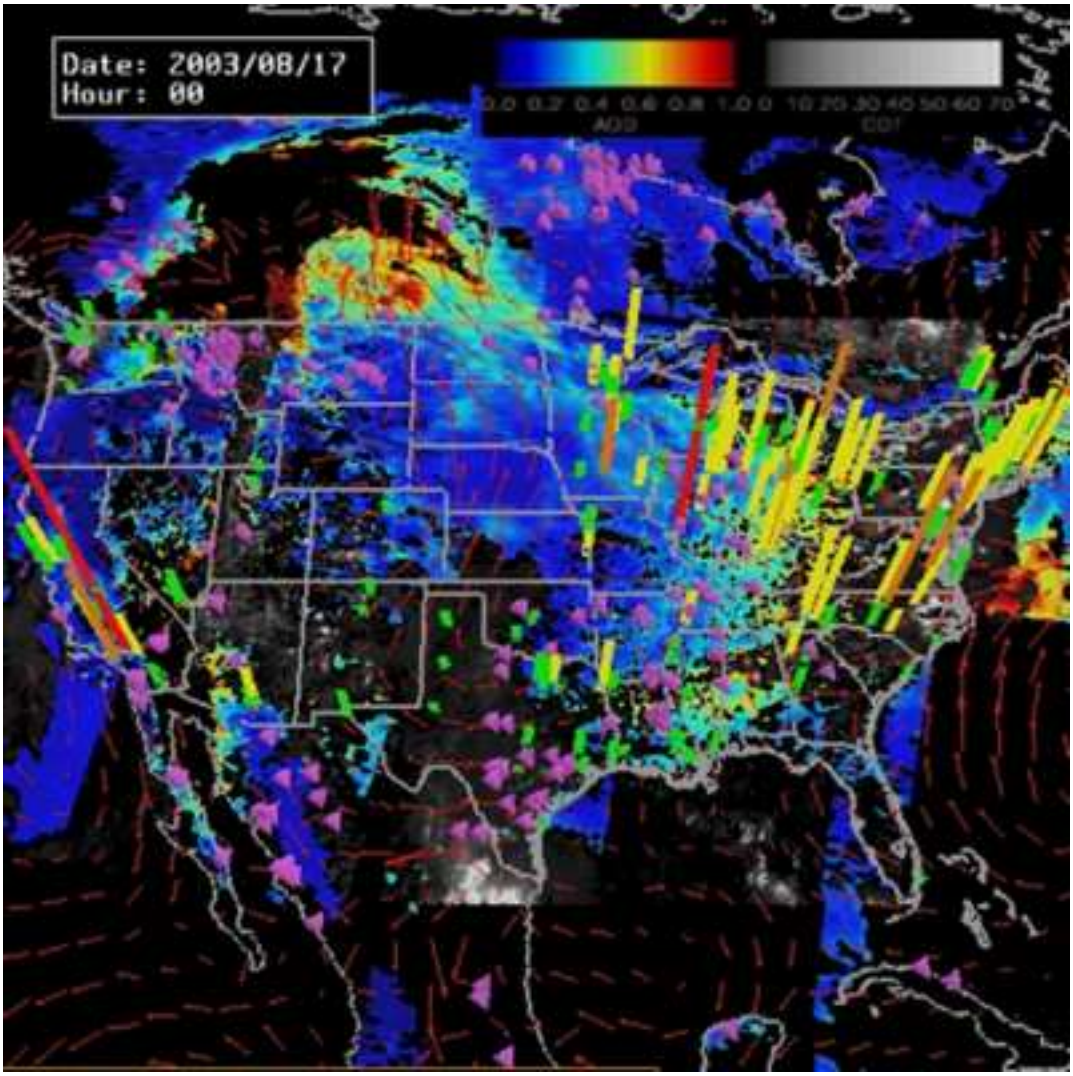
1.1.4 Zastosowania

Wyznaczanie przepływu optycznego znajduje bardzo szerokie zastosowanie w dzisiejszych czasach. Jest ono wykorzystywane nie tylko do wykrywania ruchu obserwatora lub badanego obiektu, ale również do analizowania struktury owego obiektu i jego otoczenia [4]. Spośród ciągle rosnącej ilości zastosowań przepływu optycznego można wydzielić trzy grupy:

- detekcja ruchu i położenia obiektów,
- kompresja video,

— tworzenie struktur trójwymiarowych.

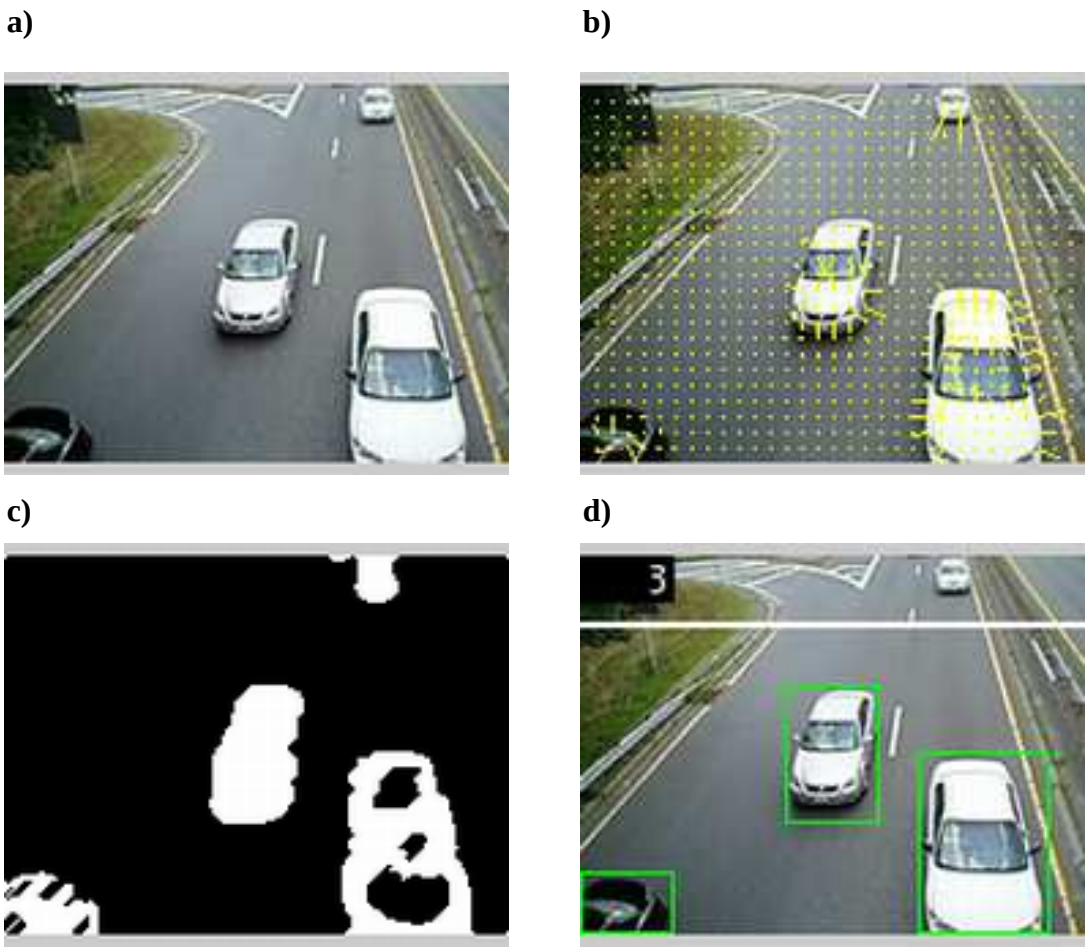
Wykrywanie ruchu stosowane jest do znajdowania poruszających się obiektów i ich trajektorii. Stosuje się je w śledzeniu zmian prądów powietrza na mapach meteorologicznych [3] czy automatycznym sterowaniu pojazdami lub robotami. Przykładową mapę pogody z naniesionymi wektorami ruchu obrazującymi m.in. zmiany pogodowe, przedstawia rysunek 1.4 :



Rys. 1.4 Mapa z naniesionymi wektorami (kolor czerwony) określającymi kierunek wiatru [6].

Detekcja przemieszczenia jest również wykorzystywana w analizie ruchu ulicznego, a także rozpoznawaniu i śledzeniu pojazdów (ang. *Tracking Cars*). Zobrazowanie działania tej metody jest możliwe m.in. za pośrednictwem narzędzia Simulink z pakietu Matlab i dzięki bibliotece *Video and Image Processing Blockset™ 2.5* [7]. Występujący w niej model, wykorzystując metody obliczania przepływu optycznego, wyszukuje wektory

ruchu. Na podstawie ich tworzy binarny obraz z naniesionymi w miejscu samochodów kształtami i porównując z obrazem rzeczywistym zaznacza samochody tam występujące. Opisane śledzenia pojazdów według kolejności przedstawionych wyżej etapów przedstawia rysunek 1.5:



Rys. 1.5 Kolejne etapy śledzenia poruszających się pojazdów [7]: a) obraz z kamery, b) wyznaczanie przepływu optycznego, c) operacje morfologiczne i binaryzacja rysunku, d) zaznaczenie znalezionych pojazdów i zliczenie ich.

Detekcja ruchu na podstawie pola wektorów znajduje również zastosowanie w celach militarnych.

Przepływ optyczny jest wykorzystywany także w kompresji video [8], np. w standardzie MPEG (ang. *Moving Picture Experts Group*). W filmach wielokrotnie występuje sytuacja, w której tło lub fragment obrazu przesuwa się. Jeżeli deseń takiego fragmentu nie zmienia się, wtedy reprezentację kolejnej klatki można przedstawić jako kombinację obrazu poprzedniego i odpowiednich wektorów ruchu. Algorytm dzieli obraz na bloki 16x16 pikseli i dla każdego z nich oblicza wektor wskazujący w miejsce

najbardziej do niego podobne w klatce poprzedniej. Do odbiornika natomiast przesyłana jest jedynie wartość wektora i różnice pomiędzy kodowanym wektorem i najbardziej do niego podobnym. Wpływa to na bardzo dużą oszczędność w ilości przesyłanych danych. Inne standardy również korzystają z podobnych metod polepszających kompresję, np. kodery H.261, H.263 lub H264 używany w HDTV.

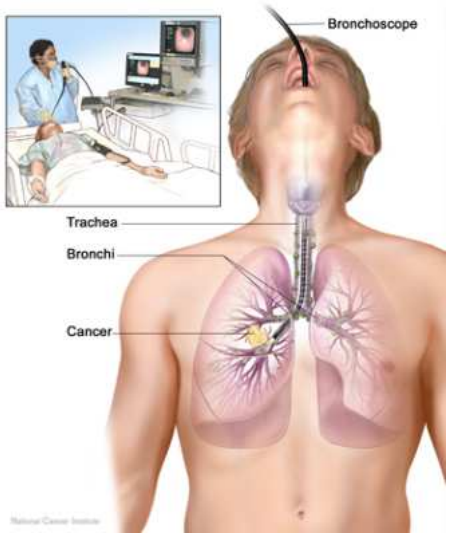
Przepływ optyczny jest również wykorzystywany do znajdowania cech obiektów 3D na podstawie zbioru płaskich, dwuwymiarowych ramek otrzymywanych w różnych chwilach czasowych obserwowanego obiektu. Ma ona zastosowanie do tworzenia efektów specjalnych lub gier komputerowych. Sztandarową pozycją kinematografii, w której użyto technikę generowania struktur 3D na podstawie ruchu (SFM, ang. *Structure For Motion*) jest trylogia *Matrix*. Zastosowano go tam w wielu scenach m.in. do modelowania twarzy bohaterów:

Rys. 1.6 Wykorzystanie przepływu optycznego do tworzenia trójwymiarowych modeli twarzy aktorów [9].

Za pomocą przepływu optycznego śledzono ruch każdego piksela dla widoków pochodzących z kilku kamer, następnie z obliczonego pola wektorów został utworzony obiekt złożony z siatki trójkątów obrazujących, np.: kształt twarzy. Na koniec wygenerowana została trójwymiarowa tekstura, ze zdjęciem twarzy aktora.

Zastosowanie technologii generacji obiektów 3D na podstawie ruchu, coraz częściej jest wykorzystywane również w medycynie. Do tego właśnie celu ma służyć algorytm testowany w tej pracy [5]. Wykorzystanie go do generacji fragmentu drzewa oskrzelowego widzianego przez kamerę bronchofiberoskopu, ma być dużym wsparciem

dla lekarza dokonującego zabiegu. Podczas takiego zabiegu głównym problemem dla lekarza jest określenie bieżącej pozycji końcówki bronchofiberoskopu w stosunku do markerów zaznaczonych na przekrojach pochodzących z tomografii komputerowej (CT). Metodą pozwalającą na automatyczne wyznaczanie pozycji w drzewie oskrzelowym może być powiązywanie jego kształtu wygenerowanego za pomocą SFM oraz otrzymanego z segmentacji danych CT. Dodatkową zaletą posiadania struktury drzewa oskrzelowego generowanego w czasie rzeczywistym jest spojrzenia na określony obszar z dowolnego kierunku i odległości.



Rys. 1.7 Widok zabiegu bronchoskopii. Bronchi – oskrzela, których struktura 3D ma być wygenerowana.

Na koniec tego podrozdziału należy dodać, że istnieją dwie główne techniki służące do generacji struktur przestrzennych w oparciu o płaskie ramki z sekwencji video. Są to *Structure From Motion* i *Stereo Vision.* W obu sposobach wykorzystuje się wiele ujęć tego samego przedmiotu, a odpowiadające sobie punkty umożliwiają obliczenie trójwymiarowego obrazu. W pierwszej metodzie wykonuje się zdjęcia w różnych chwilach czasu, zakładając, że obiekt wtedy nie zmienia swoich parametrów, a w drugiej, w tym samym czasie, ale z różnych punktów w przestrzeni.

W kwestii wyszukiwania wektorów ruchu należy zwrócić uwagę na jeden ważny aspekt. W wielu zastosowaniach estymowanie z rozdzielczością do jednego piksela jest nie wystarczające, np. w SFM, precyzyjnych pomiarach przemysłowych. Pomiędzy analizowanymi ramkami w omawianym przypadku występują wektory ruchu o długości

ułamkowych części piksela. Dlatego w prezentowanych obliczeniach i analizach zastosowano dokładność pod-pikselową.

1.2 Rola Interpolacji

W rzeczywistym przykładzie (analiza sekwencji zdjęć cyfrowych) wektory ruchu mają długości wyrażone liczbami rzeczywistymi, a nie całkowitymi. Z tego względu przy obliczaniu każdego z nich niezbędne jest interpolowanie obrazu. Interpolacja jakiejkolwiek funkcji matematycznej ma na celu wyznaczanie jej wartości pomiędzy punktami, w których jest dana. Jest to proces, który bardzo znacząco zwiększa złożoność obliczeniową algorytmu, ponieważ musi on być stosowany wielokrotnie do każdego obliczanego wektora. W tym projekcie konieczne jest otrzymanie odpowiedniej dokładności, przez co owa procedura musi charakteryzować się precyzyjnością, a ze względu na bardzo dużą ilość jej wywołań musi być również odpowiednio szybka.

Metody interpolacji obrazów działają na zasadzie dwuwymiarowej filtracji. Kolejne grupy pikseli są wymnażane ze specjalną maską współczynników wagowych, a następnie są sumowane, dzięki czemu na wyjściu pojawiają się pojedynczo wartości nowych pikseli. Jest to bardzo przydatna właściwość, gdyż piksele badanych obrazów cechują różne przesunięcia pomiędzy dwoma klatkami w sekwencji video. Z tego powodu możemy interpolować o inne przesunięcia poszczególne fragmenty, a nie całe obrazy. Każdy etap interpolacji składa się z wyznaczenia współczynników maski – jądra interpolacji dla pożądanych przesunięć, a następnie uśredniania algebraicznego z poziomami jasności pikseli obrazu znajdującymi się pod maską. W procesie interpolacji zawsze występuje punkt bazowy, posiadający wartość równą części całkowitej poszukiwanego położenia piksela. Maskę stosowaną w interpolacji z zaznaczonym na czerwono miejscem występowania tego charakterystycznego punktu przedstawia rysunek 1.8. Środkowe piksele maski to obszar w którym znajduje się poszukiwana wartość piksela. Większość metod interpolacji korzysta z podobnego schematu obliczeń. Różnią się jedynie wartościami obliczonych współczynników wagowych. Szacowane są poprzez pomnożenie odpowiadających sobie współczynników wyznaczonych dla każdego z kierunków. Następnie na tak przygotowanej masce można przeprowadzać sumowanie z funkcją obrazu:

$$O(x,y)=\sum_{i=-1}^{2}\sum_{j=-1}^{2}W(i+1,j+1)I(k+i,l+j) \tag{12}$$

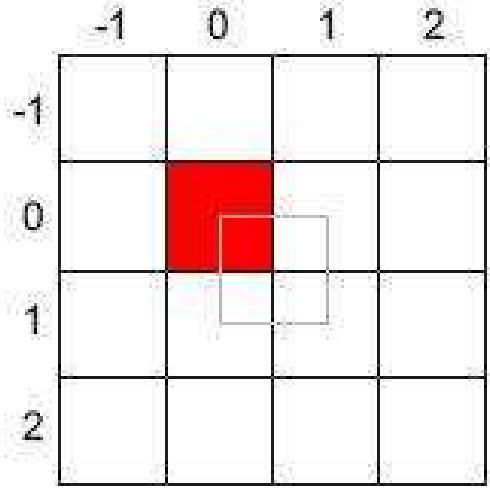
gdzie:

$O(x, y)$ – jasność wyjściowego piksela w punkcie o współrzędnych (*x*,*y*),

$W(i+1, j+1)$ – współczynniki maski 4x4 pikseli,

$I(k,l)$ – jasność wejściowego piksela w punkcie bazowym (*k*,*l*),

i, j – indeksy maski przedstawionej na rysunku 1.8.



Rys. 1.8 Maska 4x4 piksele z zaznaczonym bazowym punktem.

W przedstawianej pracy zastosowana została interpolacji bikubiczna (dwusześcienna), zwana w literaturze jako ‘cubic’ bądź ‘bicubic’. Jest to jedna z lepszych i prostszych w implementacji metod, przez co znajduje szerokie zastosowanie w wielu dziedzinach. W literaturze istnieje wiele różnych rodzajów interpolacji bikubicznej. Różnią się one głównie konstrukcją jądra interpolacji, co wyraża się w rozbieżnościach w złożoności obliczeniowej i dokładności. W prezentowanej pracy przetestowano kilka metod. Jako najbardziej precyzyjne, a równocześnie najbardziej odpowiadające wzorcowi ‘cubic’ wykorzystywanym przez Matlaba, wybrano podejście prezentowany przez Roberta G. Keys'a [10]. Przedstawione przez niego jądro interpolacji dla maski wielkości 4x4 piksele wygląda następująco:

$$u(s) = \begin{cases} (a+2)|s|^3 - (a+3)|s|^2 + 1 & 0 < |s| < 1 \\ a|s|^3 - 5a|s|^2 + 8a|s| - 4a & 1 < |s| < 2 \\ 0 & 2 < |s| \end{cases} \qquad (13)$$

$$s = x - x_k$$

gdzie:

x – punkt, w którym poszukujemy wartości,

x_k – k-ty węzeł jądra interpolacji,

s – odległość pomiędzy poszukiwanym punktem, a k-tym węzłem jądra,

a – parametr,

$u(s)$ – wartość kernela w zależności od s.

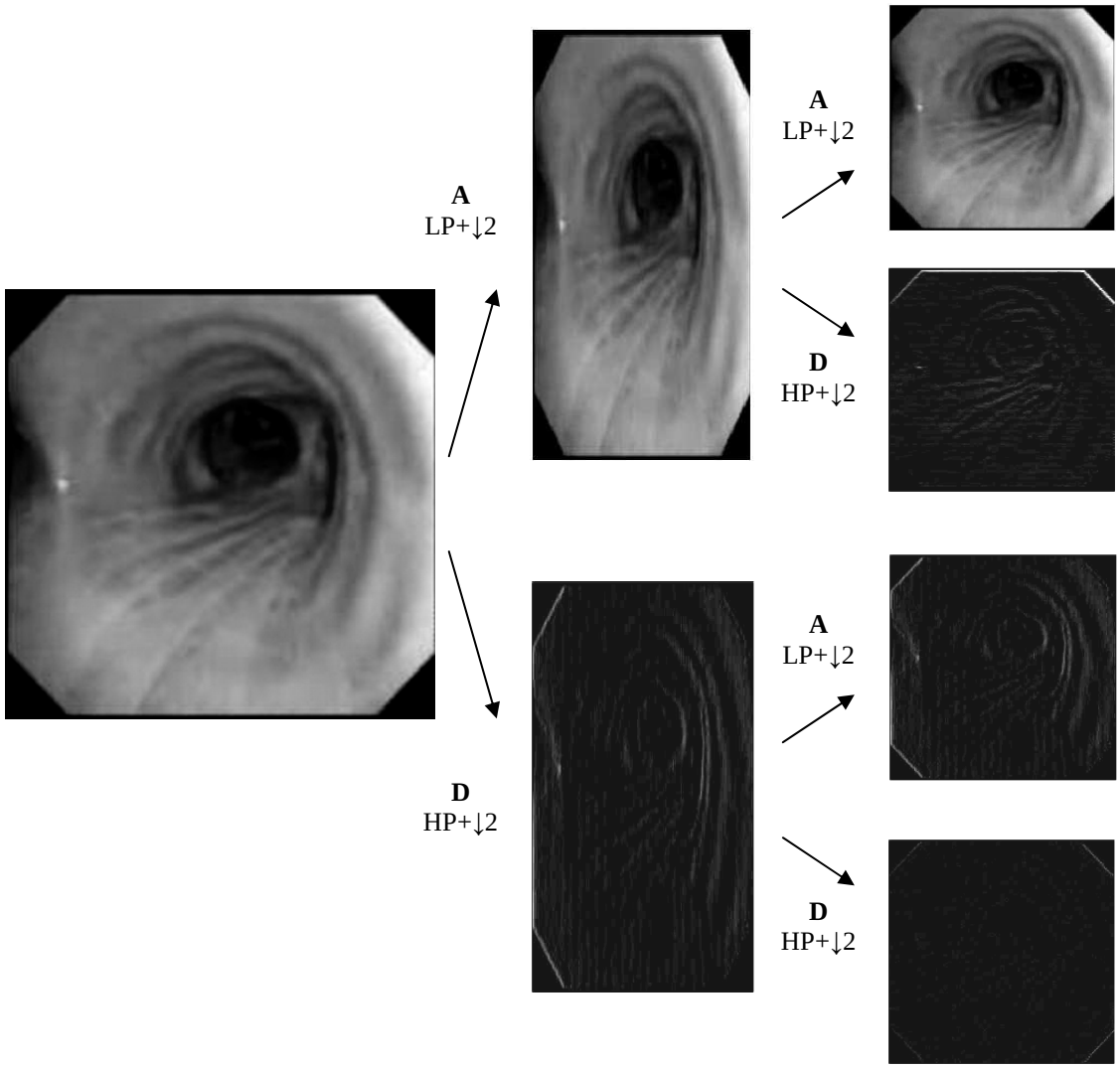
W zależności od przeznaczenie, parametr występujący w równaniach może przyjmować różne wartości. Dla osiągnięcia najbardziej wygładzonych rezultatów często zalecany jest dobór $a = -0.5$, w tym przypadku zależność (13) przybliża postać:

$$u(s) = \begin{cases} \frac{3}{2}|s|^3 - \frac{5}{2}|s|^2 + 1 & 0 < |s| < 1 \\ -\frac{1}{2}|s|^3 + \frac{5}{2}|s|^2 - 4|s| + 2 & 1 < |s| < 2 \\ 0 & 2 < |s| \end{cases} \tag{14}$$

1.3 Dekompozycja falkowa obrazu

Transformata falkowa pozwala na rozłożenie sygnału na składowe: aproksymację i zawierające szczegóły. Jest z nią związane pojęcie tzw. analizy wielorozdzielczej (ang. multiresolution analysis), która polega na wielopoziomowej reprezentacji sygnału jako sumy składowych: nisko i wysokoczęstotliwościowych. Na każdym kolejnym poziomie następuje dekompozycja składowej z poprzedniego poziomu, najczęściej jest to składowa niskoczęstotliwościowa. Schemat ten z powodzeniem wykorzystywany jest w dziedzinie przetwarzania obrazów [11]. Transformata falkowa jest tam używana głównie do kompresji obrazów i ich odszumiania. W tej pracy dekompozycję falkową wykorzystano jako narzędzie przyspieszające obliczenia. Świat obrazów jest dwuwymiarowy, więc najczęściej stosuje się w nim złożenie dwóch transformat jednowymiarowych do dekompozycji wierszy i kolumn. Ten proces działania za pomocą jednowymiarowej transformaty przedstawiono na rysunku 1.9. Po dekompozycji otrzymuje się cztery macierze współczynników, w sumie zawierające tyle samo elementów co obraz wejściowy. Dla zastosowania opisanej metody w tej pracy znaczenie ma tylko jeden

wynikowy obraz. Jest to znajdujący się na rysunku w prawym górny rogu obrazek aproksymacji – najbardziej podobny do oryginalnego. Proces dekompozycji jak wskazują symbole „↓2” na poniższym rysunku polega na decymacji przez dwa, czyli wybieraniu co drugiej próbki z sygnału. Wcześniej jednak zachodzi filtracja dolno- bądź górno-pasmowa.



Rys. 1.9 Schemat dekompozycji obrazu, pochodzącego z kamery bronchoskopu, za pomocą jednowymiarowej transformaty falkowej. Obrazek pochodzi z [12].

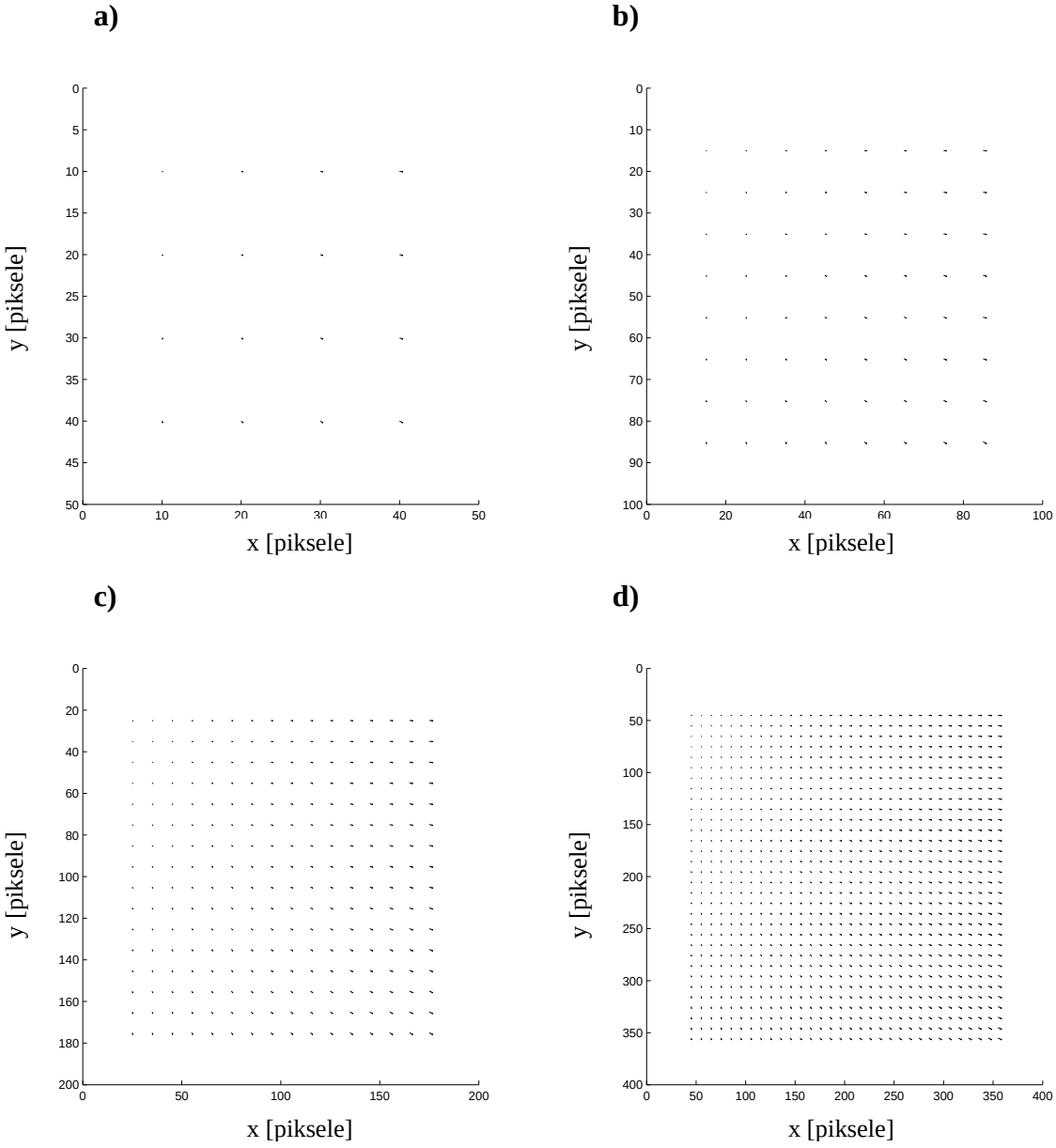
Zasada wykorzystania algorytmu dekompozycji opiera się na zastosowanie go do obu porównywanych obrazów w procesie wyszukiwania wektorów ruchu. Wtedy wyszukiwanie jest dużo szybsze, ponieważ odbywa się na cztery razy mniejszych obrazkach, więc estymuje się tyle razy mniej przemieszczeń. Tak znaleziony każdy wektor przy przejściu o poziom niżej, zostaje odpowiednio przeskalowany i zastępowany przez cztery wektory. Wtedy dokonywana jest jedynie korekcja obliczonych przesunięć. W celu uzyskania jak najlepszej efektywności i szybkości działania, dokonuje się dekompozycji na

kilku poziomach. W prezentowanej pracy zastosowane są trzy poziomy dekompozycji obrazu. Takie hierarchiczne podejście przedstawia rysunek 1.11. Został on rozjaśniony dla lepszej widoczności szczegółów obrazu.

Rys. 1.10 Trzypoziomowa dekompozycja testowanego obrazu.

Dodatkowym przyspieszeniem w znajdowaniu przepływu optycznego, przy używaniu falkowej analizy wielorozdzielczej, jest zastosowanie różnej liczby wektorów dla poszczególnych poziomów. W opisywanej pracy zaproponowana została siatka 4x4 wektorów na trzecim poziomie dekompozycji. Następnie przechodząc do niższych poziomów, każdy wektor zostaje przeskalowany i zastąpiony przez cztery identyczne wektory, ale przesunięte między sobą tak, aby ostatecznie stworzyły nową jednorodną siatkę, wykorzystywaną na niższym poziomie dekompozycji. Takie podejście jest bardzo dobre gdyż pozwala na każdym poziomie jedynie nieznacznie modyfikować wektory wygenerowane poziom wyżej. Można powiedzieć, że wektory liczone „od zera” są jedynie na poziomie najwyższym (trzecim), a następnie występuje tylko ich korekcja (dużo

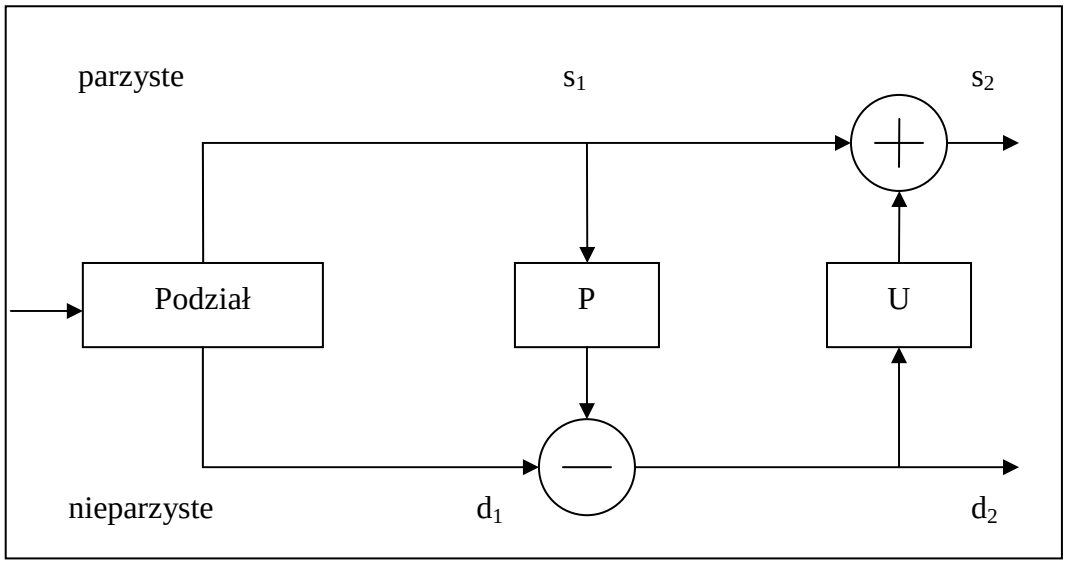
mniejszy obszar poszukiwań). Pola wektorów ruchu na różnych poziomach dekompozycji przedstawia rysunek 1.11.



Rys. 1.11 Przepływ optyczny na różnych poziomach dekompozycji: a) trzeci poziom – siatka 4x4 piksele, b) drugi poziom – siatka 8x8 pikseli, c) pierwszy poziom – siatka 16x16 pikseli, d) obraz oryginalny – siatka 32x32 piksele.

Wyżej opisana dekompozycja opiera się na filtracji i decymacji. Nie musi odbywać się, więc jedynie za pomocą falek. Jednym z prostych przykładów dekompozycji obrazu jest uśrednianie bloków 2x2 i na tej podstawie wybieranie jednego piksela do nowego obrazu na kolejnym poziomie dekompozycji. Wskutek takiego prostego schematu działania sposób ten jest mniej złożony obliczeniowo niż przykładowa filtracja za pomocą falek, ale

równocześnie mniej precyzyjny. Z tego względu, aby algorytmy były efektywne pod względem czasowym, do celów dekompozycji została zastosowana szybka predykcyjna transformata falkowa (ang. *Lifting Wavelet Transformation*). Schemat działania przedstawia rysunek 1.12.



Rys. 1.12 Schemat prostej predykcyjnej transformaty falkowej.

Symbole na rysunku:

P – blok predykcji,

U – blok uaktualnienia,

Podział – selekcja parzystych i nieparzystych próbek,

s – współczynniki aproksymacji,

d – współczynniki detali.

1.4 Miary podobieństwa

Opisana w rozdziale 1.1 korelacyjna metoda obliczania przepływu optycznego, która jest stosowana w tej pracy, polega na analizowaniu odpowiedniości poszczególnych fragmentów obrazu i na tej podstawie estymuje wektory ruchu. W celu porównania określonych bloków można stosować wiele kryteriów. Wybór odpowiedniego jest bardzo ważny, szczególnie przy korzystanie z metod gradientowych, gdzie minimalizowaną funkcją, jest właśnie procedura określająca miarę podobieństwa bloków. Najbardziej znane spośród nich to suma lub średnia bezwzględnych różnic pomiędzy wartościami odpowiednich pikseli bloków. Pierwsze z wymienionych kryteriów przedstawia (17):

$$J = \sum_{y=1}^{k} \sum_{x=1}^{l} \left(p1(x, y) - p2(x, y) \right) \tag{15}$$

gdzie:

p1 – blok wycięty z obrazu pierwszego,

p2– blok wycięty z obrazu drugiego,

k,l – wymiary bloków,

x,y –współrzędne porównywanych pikseli w bloku.

Kryterium to jest prostsze do obliczeń, gdyż zajmuje mniej cykli zegara, jednak charakteryzuje się również gorszą dokładnością. Zapewniającym mniejszy błąd estymacji wektorów ruchu, jest sposób wykorzystujący sumę kwadratów różnic jasności poszczególnych pikseli. Ten proces jest bardziej czasochłonny ze względu na konieczne dodatkowe mnożenie dla każdego elementu porównywanych bloków:

$$J = \sum_{y=1}^{k} \sum_{x=1}^{l} \left(p1(x, y) - p2(x, y) \right)^2 \tag{16}$$

Metoda (18) jest wykorzystywana w omawianej pracy. W niektórych projektach stosowane jest również pierwiastkowanie powyższego kryterium, zwiększa to jednak znacznie czas obliczeń i nie jest używane w aplikacjach, gdzie szybkość wykonywania programu jest rzeczą ważną. Jeżeli miarę odległości używa się jedynie do porównywania, wtedy pierwiastkowanie jako funkcja monotoniczna może być pominięte, ponieważ nie zmienia wyników.

1.5 Przygotowanie eksperymentu

Przed rozpoczęciem wyszukiwania wektorów ruchu należy przygotować obrazki, które będą testowane. Bardzo ważne jest aby były one przygotowane w sposób umożliwiający sprawdzanie różnych cech algorytmów. Docelowo algorytm powinien porównywać kolejne klatki w sekwencji video dlatego obrazki wybrane do testów powinny różnić się nieznacznie. Warty podkreślenia jest fakt, że nie możemy do tego celu użyć dwóch obrazków rzeczywistych, chociaż na takich działają zwykle finalne wersje programów. Podczas testowania algorytmów zawsze konieczne jest odniesienie wyników pomiarów do istniejącego wzorca. W tym przypadku wzorcem jest znane pole wektorów ruchu, względem którego porównywane będą rezultaty obliczeń. W celu jego otrzymania został użyty obraz rzeczywisty, przekształcony w określony sposób. Jako najbardziej adekwatny do przeznaczenia, został użyty obraz pochodzący z kamery bronchoskopu. Drugi obraz otrzymano poprzez rozciągnięcie pierwszego za pomocą funkcji interpolującej

Matlaba. Aby uzyskać niewielkie przesunięcia, takie jak występują pomiędzy obrazami w sekwencji pochodzącej z kamery, dokonano interpolacji w punktach różniących się od oryginalnych o setne części piksela.

Interpolacja została wykonana przy użyciu procedury **interp2** przeznaczonej dla funkcji dwuwymiarowych. Jako metodę wybrano ‘cubic’ czyli najczęściej stosowaną i dającą dobre efekty interpolację bikubiczną. Dane które posłużyły w procesie interpolacji wykorzystane zostały również do stworzenia wzorcowych wektorów ruchu – powstałych wskutek przekształcenia obrazu pierwszego w drugi. Współrzędne dx i dy wektorów prędkości (1) zostały obliczone następująco:

$$dx = x - x \cdot ix \tag{17}$$

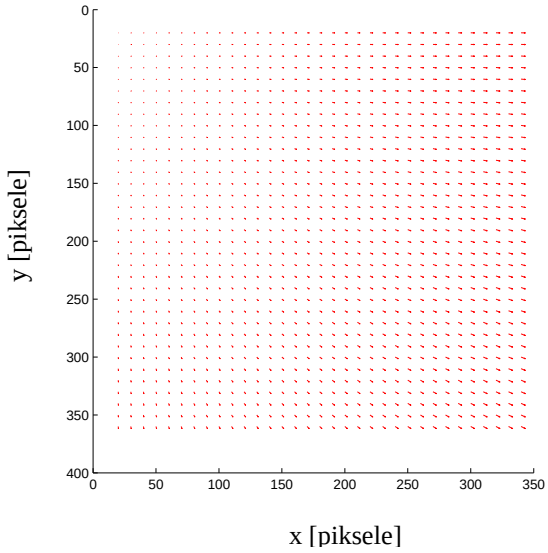
$$dy = y - y \cdot iy$$

gdzie:

x, y – współrzędne punków obrazka wzorcowego,

ix – wartość co którą znajdowany jest nowy punkt wzdłuż osi X (0,99),

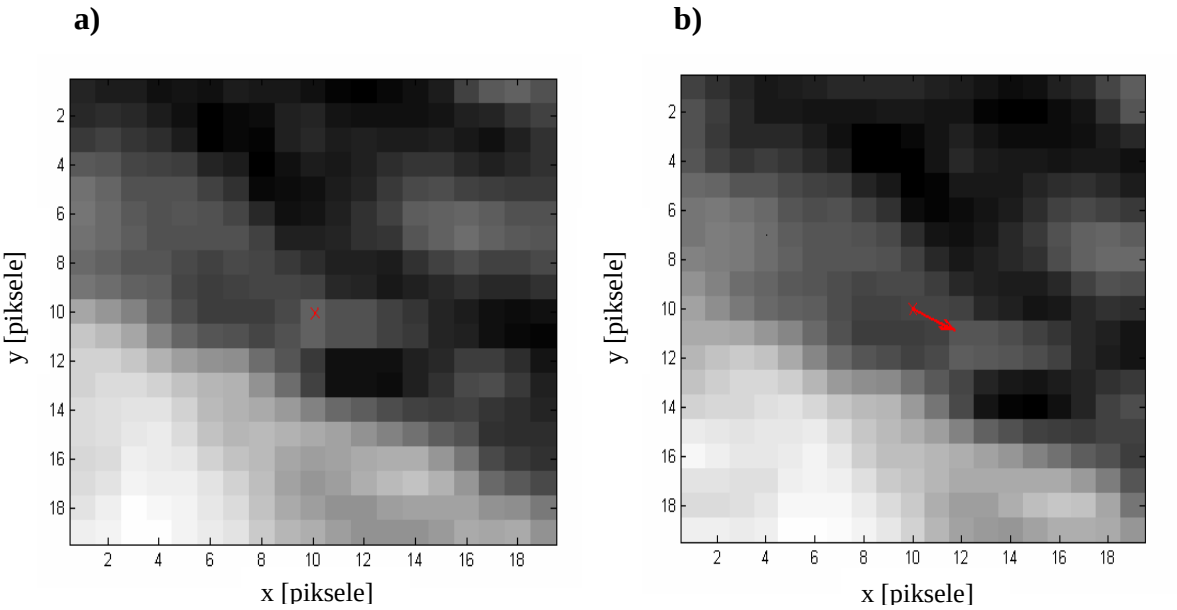
iy – wartość co którą znajdowany jest nowy punkt wzdłuż osi Y (0,995).



Rys. 1.13 Obrazek oryginalny wraz z naniesionym polem wzorcowych wektorów ruchu.

Wygenerowaną siatką wzorcowych przesunięć przedstawia rysunek 1.13. Punkty zaczepienia wektorów wzorcowych były umiejscowione co 10 pikseli, co dało ich łączną ilość równą 1155. Konieczne jest stosowanie dużej liczby wektorów zamiast jednego ze względu na właściwości kamery bronchoskopu. Posiada ona szerokokątny obiektyw co powoduje zniekształcenia geometryczne widzianych przedmiotów. Wskutek tego niektóre obszary mogą wykazywać większe, a inne – mniejsze przesunięcia. Dla przykładu, na

rysunkach 1.14a i 1.14b, prezentowane są fragmenty obrazu (18x18 pikseli) wraz z przedstawionym wektorem ruchu, obrazującym zmianę położenia punktu na obrazie.



Rys. 1.14 Fragmenty obrazu z zaznaczonym punktem podlegającym przekształceniu: a) obraz wzorcowy, b) obraz po przemieszczeniu.

2 Algorytmy wyznaczania przepływu optycznego

W rozdziale drugim opisane zostaną, wykorzystywane w części autorskiej, różne sposoby obliczania pola wektorów ruchu. Na początku przedstawiono główny algorytm wyznaczania przepływu optycznego z wyszukiwaniem siłowym. Następnie zaprezentowane zostaną inne zoptymalizowane algorytmy: gradientowa metoda najszybszego spadku (ang. *Steepest Descent*), a także podejście bezgradientowe – algorytm sympleksowy Neldera-Meada. Jako uzupełnienie rozdziału opisano grupę metod, wykorzystujących wyłącznie siatkę regularną.

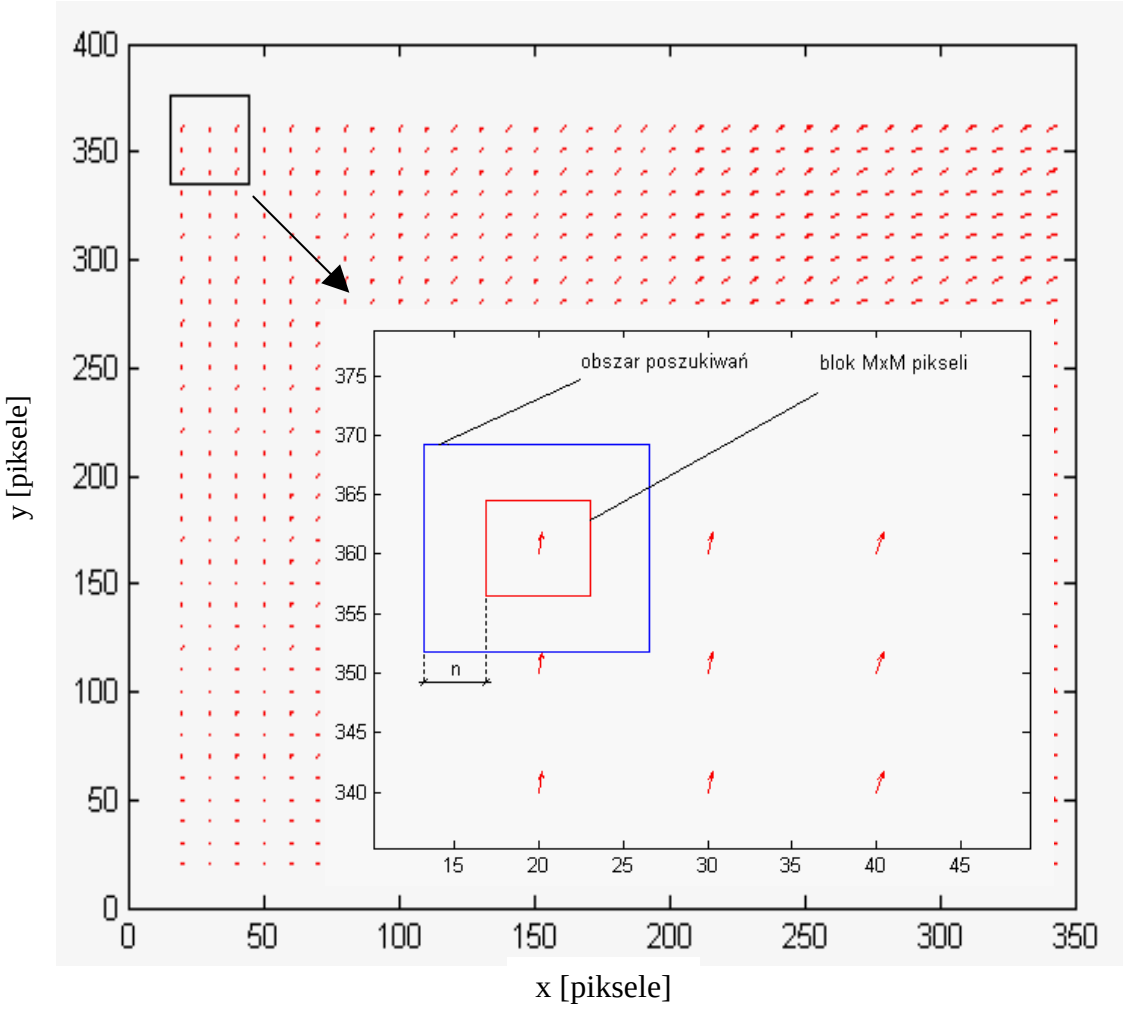
2.1 Przeszukiwanie siłowe

Bardzo często w zastosowaniach, gdzie potrzebna jest pewność generowanych wyników, wykorzystywane jest siłowe wyszukiwanie wektorów. Jest to algorytm typu *Brute force* („brutalna siła” [13]), który w poszukiwaniu rozwiązania problemu sukcesywnie sprawdza wszystkie możliwe kombinacje, zamiast skupić się na optymalizacji zagadnienia. Takie podejście zwiększa znacznie czas obliczeń. Biorąc pod uwagę, że w pracy stosowana jest dokładność pod-pikselowa, można wstępnie wyobrazić sobie jak bardzo złożony staje się program, w którym dla znalezienia jednego wektora konieczna jest seria kilku tysięcy interpolacji. Bezsprzecznie jednak metoda siłowa jest najbardziej stabilna i o ile określony zostanie poprawny obszar poszukiwań (odpowiednio duży), znajduje ona precyzyjne wartości wektorów ruchu. Na uwagę również zasługuje fakt, że metoda ta, przy dobrym ustawieniu parametrów algorytmu, znajduje zawsze minima globalne. Celem tej pracy jest zaproponowanie metod znajdujących wektory ruchu z dokładnością metody siłowej, jednak w dużo krótszym czasie.

Algorytm siłowy (ang. *Exhaustive Search*) działa na zasadzie przesuwania zaznaczenia grupy pikseli na jednym z obrazów i porównywania jej z określonym fragmentem tej samej wielkości na obrazie drugim. Zakres sprawdzanych przemieszczeń bloku z obrazu pierwszego zależy od wybranego obszaru poszukiwań. Metoda ta sprawdza z zadaną rozdzielczością dla jakiego przesunięcia, funkcja podobieństwa pomiędzy poszczególnymi blokami obrazów, przyjmuje wartość najmniejszą. Wiąże się to z wykonaniem tysięcy interpolacji i porównań dla każdego obliczanego wektora. Oznaczenia i pojęcia związane z metodą siłową przedstawiono poniżej:

— i1, i2 – obraz pierwszy i drugi, o rozmiarze x = 1…X, y = 1…Y,
— p1, p2 – bloki MxM pikseli wycięte odpowiednio z pierwszego i drugiego obrazu,
— n – liczba określająca połowę obszaru poszukiwań,
— r – rozdzielczość – częstotliwość próbkowania obrazu, wartość najmniejszego możliwego przesunięcia przy poszukiwaniu wektorów ruchu,
— m – struktura zawierająca współrzędne początku: x, y (wielkości wejściowe) i wartości przesunięcia: dx, dy (wielkości wyjściowe) jednego wektora ruchu,
— s – tablica zawierająca wszystkie przemieszczenia dx, dy możliwe w obszarze poszukiwań,
— L – ilość par przesunięć dx i dy w s,
— K – ilość wektorów ruchu.

Opisane powyżej pojęcia bloku i obszaru poszukiwań, są przedstawione na rysunku 2.1.



Rys. 2.1 Wyjaśnienie pojęć stosowanych w metodzie siłowej obliczania przepływu optycznego.

Poza wielkością obszaru poszukiwań, na wyniki otrzymane w metodzie siłowej (jak również innych prezentowanych w tej pracy, ponieważ postępowanie „blokowe" jest stosowane w każdej omawianej tutaj metodzie – przyp. autora) wpływa wielkość stosowanych bloków pikseli. Im większy blok tym mniejszy szum, który wpływa niekorzystnie na wyniki estymacji. Mimo to, równocześnie zwiększa się ryzyko błędnego skorelowania obszarów i mniej precyzyjnego wyliczenia wektorów. Wynika z tego, że rozmiar stosowanych bloków musi być kompromisem pomiędzy kompensacją szumu i dokładnym podobieństwem odpowiednich grup pikseli.

Jeszcze jednym ważnym elementem wpływającym na wyniki w przeszukiwaniu siłowym jest rozdzielczość z jaką próbkuje się obraz. Według przewidywań, im większa rozdzielczość, tym dokładniejsze wyniki estymacji, istnieje jednak pewna granica, powyżej której dokładność polepsza się bardzo nieznacznie. Szczegółowe wyniki przedstawione są w rozdziale czwartym. Ponadto zwiększanie częstotliwości próbkowania powoduje gwałtowny wzrost, i tak długiego już, czasu obliczeń.

Schemat postępowania w wyszukiwaniu wektorów ruchu metodą siłową przedstawiono poniżej. Jest to wzorzec wykorzystywany w części implementacyjnej programu.

- Inicjalizacja obrazów i1 i i2, częstotliwości r oraz siatki początków wektorów ruchu,
- Dla wektorów od 1 do K:
  - Pobranie współrzędnych początku kolejnego wektora ruchu i zapisanie ich do m,
  - Z i2 wycięcie bloku p2, rozpoczynając od współrzędnych wektora m,
  - Dla zadanego obszaru poszukiwań [-n,n], wygenerowanie wektora wszystkich możliwych przemieszczeń s,
  - Dla każdej pary przesunięć z wektora s, i = 1…L:
    - Wyznaczenie wierzchołka p1 jako położenia po przemieszczeniu z początku m o wektor s(i),
    - Używając interpolacji z i1 wycięcie p1,
    - Obliczenie i zapis sumy kwadratów różnic pomiędzy p1 i p2,
  - Wybór bloku o najmniejszej wartości funkcji celu i podstawienie współrzędnych przesunięcia do współrzędnych prędkości wektora m,
- Zapis wszystkich wyliczonych wartości wektorów do pliku.

Podsumowując w przeszukiwaniu siłowym używanym do wyznaczania przepływu optycznego ważne są trzy czynniki:

- rozmiar obszaru poszukiwań,
- wielkość porównywanych bloków,
- częstotliwość – rozdzielczość z jaką badany jest przepływ.

W informatyce omawianą metodę siłową stosuje się ze względu na dużą prostotę w implementacji. Szczególnie wykorzystywana jest tam gdzie prędkość obliczeń jest mniej ważna, za to wystąpienie błędu może mieć poważne konsekwencje. Poza określeniami wspomnianymi powyżej, algorytm przeszukiwania siłowego występuje również pod nazwą GT (ang. *Generate and Test*).

2.2 Metoda najszybszego spadku

Omawiana w tym rozdziale metoda należy do gradientowych sposobów minimalizacji funkcji wielowymiarowych. Stosowana jest ona, podobnie jak algorytm siłowy, do rozwiązywania wielu matematycznych problemów. Metody gradientowe polegają na wyznaczaniu kolejnych kierunków poszukiwań, bazując na znajomości gradientu funkcji celu, w punkcie osiągniętym w poprzednim kroku algorytmu. Prezentowana w tym rozdziale strategia najszybszego spadku jest jedną z najprostszych, a równocześnie najpopularniejszych metod, wykorzystujących obliczanie pochodnych przy znajdowaniu minimum.

W omawianym algorytmie w trakcie kolejnych iteracji wywoływane są dwie procedury: licząca gradient i wartość funkcji celu. Jest to metoda optymalizacji bez ograniczeń, co oznacza, że algorytm poszukuje rozwiązania w całej dziedzinie funkcji. Każda kolejna pętla algorytmu polega na znalezieniu nowego kierunku poszukiwań, który jest wskazywany przez ujemny gradient. Długość następnie wykonywanego kroku może być stała lub zmienna w sposób zapewniający znajdowanie minimum funkcji w wyznaczonym kierunku. Drugi z tych sposób jest stosowany w metodzie najszybszego spadku. Wykorzystanie jej znacznie przyspiesza czas znajdowania wektorów ruchu, gdyż ilość koniecznych interpolacji jest wielokrotnie mniejsza niż w opisywanej wcześniej metodzie. Z tego względu postępowanie to może być dobrym sposobem na optymalizacje algorytmu, jednak metoda ta, jak i inne metody gradientowe nie zapewniają osiągnięcia minimum globalnego – w zależności od punktu startowego oraz właściwości minimalizowanej funkcji celu istnieje ryzyko zakończenia algorytmu w minimum

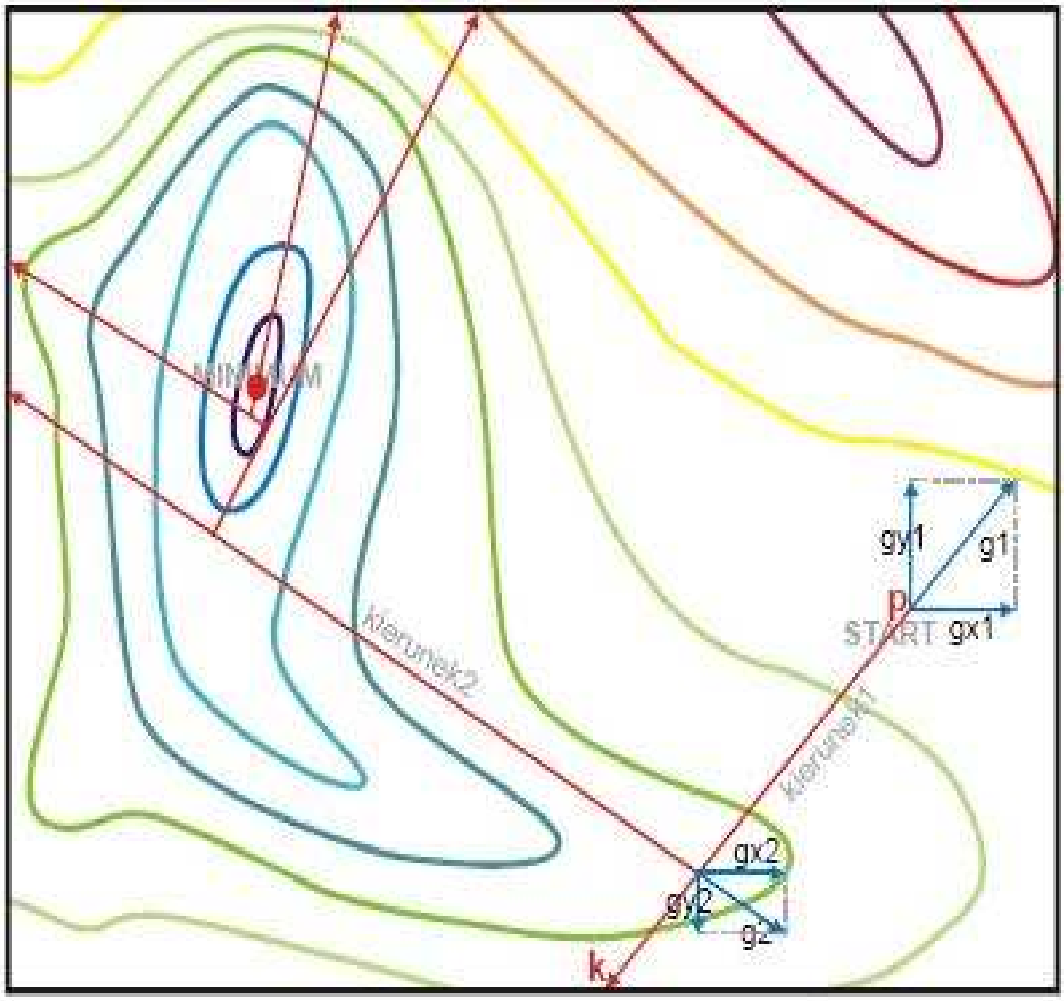
lokalnym. Różnica w porównaniu z metodą siłową tkwi w braku pojęcia obszaru poszukiwań. Kolejne przesunięcia p1 są otrzymywane na podstawie następnych kroków algorytmu zmierzającego do minimum, a nie sztywnego sprawdzania wielu różnych przemieszczeń. W tego typu metodach ważny jest warunek zakończenia obliczeń. Z reguły stosowane sposoby na przerwanie działania programu to: wykonanie liczby zadanych iteracji, spadek wartości gradientu poniżej ustalonej wartości lub wystarczająca mała wartość minimalizowanej funkcji celu. W opisywanej pracy zaimplementowany jest pierwszy i trzeci z wymienionych sposobów. Schemat działania algorytmu opisany jest poniżej i przedstawiony rysunku 2.2 [14].

1. Krok i=0, przyjęcie punktu startowego (x_i, y_i).
2. Wyznaczenie w punkcie (x_i, y_i) gradientu $\vec{g}(x_i, y_i)$.
3. Jako kierunek poszukiwania minimum przyjmuje się - $\vec{g}(x_i, y_i)$.
4. Sprawdzenie czy $\|\vec{g}(x_i, y_i)\| < \varepsilon$, gdzie ε to liczba bliska zera.
5. Wykonywanie minimalizacji kierunkowej w wyznaczonym kierunku.
6. Krok i=i+1, znalezione minimum to nowy punkt (x_i, y_i).
7. Powrót do punktu 2.

W prezentowanym sposobie ważny jest fragment dotyczący minimalizacji funkcji w danym kierunku. Polega on na wykorzystaniu metod jednowymiarowych. Spośród wielu przykładów rozwiązywania tego problemu jednym z najbardziej popularnych jest schemat korzystający ze złotego podziału odcinka. Nie może być on jednak zastosowany w omawianej pracy, ponieważ wykorzystuje się go jedynie w przypadkach posiadających tylko jedno minimum. Bardziej skomplikowanym podejściem, które zostało zaimplementowane w tej pracy, jest postępowanie oparte o metodę Brent'a. Każdy krok metody zapamiętuje sześć punktów – a, b, u, v, w, x:

- (a,b) to przedział w którym leży minimum,
- x to najlepsze przybliżenie minimum,
- w to druga co do wartości najmniejsza wartość,
- v jest poprzednią wartością w,
- u jest punktem w którym wartość funkcji była liczona ostatnio.

Bardziej szczegółowe informacje na temat algorytmu najszybszego spadku zostały przedstawione w [14].



Rys. 2.2 Przykład metody najszybszego spadku użyty dla funkcji o jednym minimum [14].

Jak wspomniano wcześniej ważne ze względu na warunek zakończenia algorytmu jest również dobranie odpowiedniej tolerancji. Określa ona wartość funkcji celu, poniżej której otrzymywane wyniki są uznane za wystarczająco dokładne i nie konieczne jest kontynuowanie obliczeń.

Wykorzystanie metody najszybszego spadku w wyszukiwaniu wektorów ruchu, przedstawiono w znajdującym się poniżej. Podobnie jak przy metodzie siłowej przedstawiony algorytm jest używany do późniejszej implementacji. Oznaczenia nie występujące wcześnie przedstawiono poniżej:

- L – ilość iteracji w algorytmie najszybszego spadku,
- $\vec{g}(x, y)$ – gradient w punkcie (x, y).
- (dx, dy) – wektor przemieszczeń charakterystycznych dla kolejnych kroków algorytmu.

- Inicjalizacja obrazów i1 i i2, częstotliwości r oraz siatki początków wektorów ruchu,
- Dla wektorów od 1 do K:
  - Pobranie współrzędnych początku kolejnego wektora ruchu i zapisanie ich do m,
  - Z i2 wycięcie bloku p2, rozpoczynając od współrzędnych wektora m,
  - Zapamiętanie początku wektora m,
  - Obliczenie wartości funkcji celu oraz jej gradientu w punkcie startowym m,
  - Dla założonej liczby iteracji od 1 do L:
    - Wykonanie kroku według algorytmu najszybszego spadku i pobranie wartości przemieszczeń dx, dy i zapisanie ich do m,
    - Sprawdzenie czy $\|\vec{g}(x_i, y_i)\| < \varepsilon$, jeżeli tak – następuje przerwanie tej pętli,
    - Wyznaczenie wierzchołka p1 poprzez przemieszczenie z początku wektora m o wektor (dx,dy),
    - Używając interpolacji z i1 wycięcie p1,
    - Obliczenie sumy kwadratów różnic pomiędzy p1 i p2, jako wartości funkcji na podstawie, której wykonywany jest następny krok,
  - Podstawienie różnicy współrzędnych początkowych wektora m i ostatnio znalezionego minimum do składowych prędkości wektora m,
- Zapis wszystkich wyliczonych wartości wektorów do pliku.

Jako podsumowanie należy podkreślić, że w wyszukiwaniu gradientowym, na jakość i szybkość obliczania przepływu optycznego wpływają:

- odpowiednie obliczanie kierunku poszukiwań,
- dokładna minimalizacja kierunkowa,
- założona tolerancja – gwarantująca zakończenie algorytmu.

2.3 Metoda sympleksów Neldera-Meada

Algorytm sympleksów Neldera-Meada (ang. *Downhill Simplex, Amoeba*), jest powszechnie używanym bezgradientowym sposobem służącym minimalizacji nieliniowych funkcji n-wymiarowych. Cechą metod bezgradientowych jest brak potrzeby

znajomości pochodnych funkcji celu, ponieważ w obliczeniach wykorzystywane są jedynie jej wartości. Często algorytmy te są nazywane metodami bezpośrednich poszukiwań (ang. *Direct Search*).

Algorytm sympleksowy to iteracyjne podejście do rozwiązywania problemów głównie w programowaniu liniowym, poprzez kolejne poprawianie rezultatów. Główną zaletą algorytmu Neldera-Meada jest jego niska złożoność obliczeniowa dla mało rozbudowanych funkcji celu. Opiera się on na budowaniu w przestrzeni wielowymiarowej tzw. sympleksu. Jest to zbiór wypukły o n+1 wierzchołkach, którego sposób tworzenia odzwierciedla poszukiwanie minimum. Celem każdej iteracji jest wymiana „najgorszego" wierzchołka i zbudowanie nowego wielościanu. Dla funkcji dwuwymiarowych sympleks jest trójkątem. Metoda polega na tym, że jeżeli znany jest pewien wierzchołek sympleksu i wartość poszukiwanej w nim funkcji, to wszystkie wierzchołki przyjmujące gorsze wartości są odrzucane. W następnym kroku przechodzi się do kolejnego wierzchołka, leżącego na tej samej krawędzi z obliczonym już punktem, jeżeli funkcja celu osiąga tam lepsze rezultaty. Zakończenie iteracji ma miejsce, gdy wartość funkcji celu w kolejnym przeglądanym wierzchołku jest tak mała, że spełnia postawione założenia. Więcej informacji na temat prezentowanego algorytmu znajduje się w [15].

Podczas trwania algorytmu i tworzenia kolejnych sympleksów wykonywane są na nich liczne operacje, mające na celu zmierzanie do poszukiwanej wartości:

Odbicie punktu P_h względem P':

$$P^* = (1+\alpha)P' - \alpha P_h \tag{18}$$

Ekspansja punktu P^* względem P':

$$P^{**}(1+\gamma)P^* - \gamma P' \tag{19}$$

Kontrakcja punktu P_h względem P':

$$P^{***}\beta P_h + (1-\beta)P' \tag{20}$$

Redukcja sympleksu:

$$P_i = (P_i + P_l)/2 \quad i = 1,2,...,n+1 \tag{21}$$

Wyjaśnienie symboli używanych w powyższych wzorach:

α – współczynnik odbicia ($\alpha > 0$),

β – współczynnik kontrakcji ($0 < \beta < 1$),

γ – współczynnik ekspansji ($\gamma > 0$),

ε – dokładność obliczeń,

n – liczba zmiennych niezależnych,

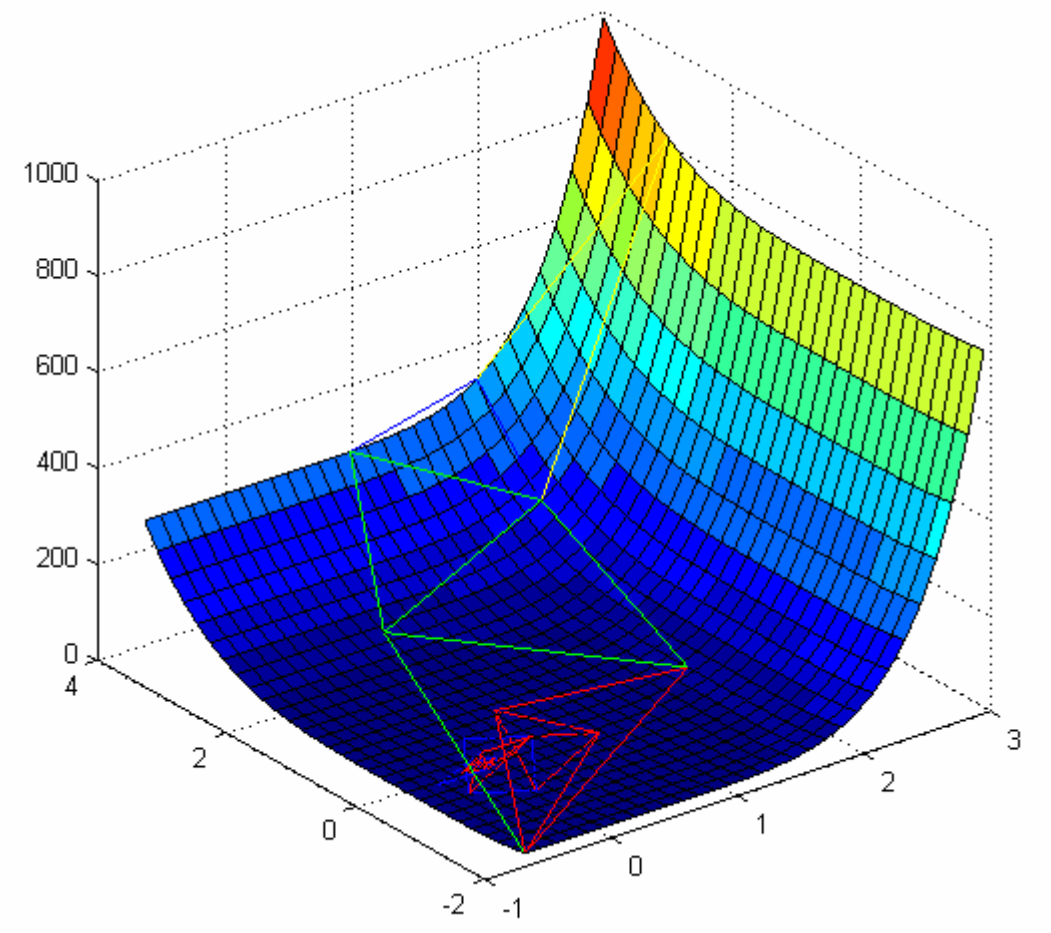
P_i – i-ty punkt simpleksu (i=1, 2, ..., n+1),

P_h – punkt sympleksu, w którym wartość funkcji osiąga największą wartość (najgorszy),

P_l – punkt sympleksu, w którym wartość funkcji osiąga najmniejszą wartość (najlepszy),

P' – środek symetrii simleksu liczony z wyłączeniem punktu P_h.

W metodzie Neldera-Meada dobiera się następujące wartości współczynników: α=1, β=0.5, γ=2. Przykładowy obraz minimalizowanej funkcji dwuwymiarowej przedstawiony jest na rysunku 2.3.



Rys. 2.3 Przykład funkcji dwuwymiarowej z przedstawionym działaniem algorytmu sympleksów Neldera-Meada [16].

Podobnie jak w metodach gradientowych ważny jest warunek zakończenia działania metody. Z reguły w algorytmie Neldera-Meada stosuje się pomiar wielkości sympleksu. W tej pracy jest to średnia długość odległości z geometrycznego środka sympleksu do jego wierzchołków. Jeżeli jest ona mniejsza niż założona tolerancja, algorytm kończy pracę. Drugim ograniczeniem, podobnie jak w poprzednich omawianych metodach, jest ilość założonych iteracji.

Schemat algorytmu obliczającego przepływ optyczny poprzez zastosowanie metody Neldera-Meada przedstawiono poniżej. Prezentowany algorytm jest używany w późniejszej implementacji. Wszystkie wykorzystywane symbole zostały już opisane wcześniej. Tym razem (dx, dy) oznacza odległość między nowym, a poprzednim wierzchołkiem sympleksu, natomiast L – jest liczbą iteracji za pomocą metody sympleksów.

- Inicjalizacja obrazów i1 i i2, częstotliwości r oraz siatki początków wektorów ruchu,
- Dla wektorów od 1 do K:
  - Pobranie współrzędnych początku kolejnego wektora ruchu i zapisanie ich do m,
  - Z i2 wycięcie bloku p2, rozpoczynając od współrzędnych wektora m,
  - Zapamiętanie początku wektora m i przyjęcie go jako punkt startowy,
  - Dla założonej liczby iteracji od 1 do L:
    - Wykonanie kroku według algorytmu Neldera-Meada, pobranie wartości przemieszczeń dx, dy do nowego wierzchołka sympleksu i zapisanie ich do m,
    - Wyznaczenie wierzchołka p1 poprzez przemieszczenie z początku wektora m o wektor (dx,dy),
    - Używając interpolacji z i1 wycięcie p1,
    - Obliczenie sumy kwadratów różnic pomiędzy p1 i p2, jako wartości funkcji w rozpatrywanym wierzchołku sympleksu,
    - Wybranie wierzchołka o najmniejszej wartości funkcji celu,
  - Podstawienie różnicy współrzędnych początkowych wektora m i ostatnio znalezionego minimum do składowych prędkości wektora m,
- Zapis wszystkich wyliczonych wartości wektorów do pliku.

Podsumowując w metodzie bezgradientowej na szybkość i dokładność rezultatów wyznaczania pola wektorów ruchu wpływają:

- skomplikowanie funkcji celu – wielowymiarowe zadania nieopłacalne numerycznie,
- założona tolerancja – gwarantująca zakończenie algorytmu.

2.4 Metody wymagające regularnej siatki

W tym rozdziale przedstawione zostaną inne sposoby estymacji przepływu optycznego, nie wykorzystywane w autorskiej części pracy. Przedstawienie ich jest uzupełnieniem przeglądu omawianych metod. Będą to metody wyznaczające wektory ruchu w oparciu o regularne siatki. Jako reprezentacja tej grupy przedstawiona zostanie metoda częstotliwościowa, bazująca na korelacji fazowej pomiędzy dwoma obrazami, a także gradientowa – wykorzystująca szybką zespoloną transformatę falkową.

2.4.1 Korelacja fazowa

Prezentację metody bazującej na porównywaniu fazy obrazów przy wyznaczaniu przepływu optycznego przedstawiono w [17]. Autor proponuje również obliczenia na blokach pikseli, reprezentujących przesunięcie danego punktu. Stosując analizę Fouriera udowadnia on, że widmo funkcji jasności piksela przed i po przesunięciu o pewien wektor, różni się jedynie częścią fazową. Następnie, znając zmianę fazy jaką wprowadza przesunięcie można obliczyć wektor ruchu.

Rozważanie to bazuje na założeniu, że tym co się zmienia pomiędzy porównywanymi fragmentami obrazów, jest część fazowa widma Fouriera, a część amplitudowa pozostaje stała. Jednakże dla małych bloków istotne zmiany mogą pojawić się, szczególnie na granicach obrazu. Z tego względu przed transformatą Fouriera stosuje się funkcję okna Gaussa.

Algorytm metody można zapisać następująco:

1. Próbkowanie obrazów z określoną częstotliwością.
2. Wybranie obszarów MxM pikseli ze środkiem w punkcie (i,j).
3. Wymnożenie bloków z funkcją okna Gaussa 2D.
4. Obliczenie ich transformat Fouriera.
5. Wyznaczenie faz.

6. Dla każdej z częstotliwości (ω_y ,ω_y) uaktualnienie przestrzeni Hougha, korzystając z (27).
7. Maksimum w transformacie Hougha określa estymacje prędkości przesunięcia badanego bloku

Omawiana metoda pozwala otrzymać dokładny i gęsty przypływ optyczny. Główne problemy mogą dotyczyć zjawisk towarzyszących transformacji Fouriera. Czynniki od których zależny jest algorytm to:

— rozmiar porównywanych bloków,

— szerokość okna czasowego.

2.4.2 Szybka, dyskretna, zespolona transformacja falkowa

Wykorzystanie transformacji falkowej do obliczania przepływu optycznego sprowadza się w tym przypadku do rozwiązywania cząstkowych równań różniczkowych opisanych przy metodach gradientowych w podrozdziale 1.1.2. Stosuje się tu właściwość szybkich numerycznych obliczeń na różnych poziomach aproksymacji. W [18] zaproponowano rodzinę falek $(\psi^s)_{s=1...S}$:

$$\psi_{jk}^{s}(x) = 2^{j}\psi^{s}\left(2^{j}x - k\right) \tag{22}$$

gdzie:

$k=(k_1, k_2)$ – indeks przesunięcia,

$x=(x_1, x_2)$ – wektor pozycji,

j – indeks skali.

Celem metody jest rozwiązanie równania przepływu optycznego przy pomocy S różnych wektorów otrzymanych dzięki zastosowaniu rodziny falek (22).

$$\left\langle I_t, \frac{\partial \psi_{jk}^{s}}{\partial x} \right\rangle u + \left\langle I_t, \frac{\partial \psi_{jk}^{s}}{\partial y} \right\rangle v = \frac{\partial}{\partial t}\left\langle I_t, \psi_{jk}^{s} \right\rangle \qquad \forall s = 1...S \tag{23}$$

Stworzony układ S równań, posiada niewiadome u i v. Zaletą metody jest możliwość obliczenia wszystkich niezbędnych współczynników za pomocą szybkiej transformacji falkowej. Dzięki temu otrzymuje się przyspieszenie obliczeń. Poza tym podany układ równań posiada mało niewiadomych, co jest dużą zaletą w porównaniu z innymi metodami. Na uwagę zasługuje fakt, że więcej jest równań niż niewiadomych, co pozwala na weryfikację poprawności obliczeń w rozważanym obszarze. Dodatkowo podejście to umożliwia rozwiązanie problemu apertury, spotykanego w metodach

gradientowych. Ponadto sposób wykorzystania transformaty falkowej umożliwia również osiągnięcie dokładności pod-pikselowej podczas obliczania przepływu optycznego.

Na jakość i szybkość obliczeń wykonywanych tą metodą wpływa głównie rodzaj użytych falek, które muszą być separowane, aby była możliwość obliczenia współczynników w równaniu (23). Opisywany algorytm może generować błędne wyniki, gdy obiekty na obrazie poruszają się z różnymi i przeciwnymi prędkościami. Wtedy konieczne jest skorzystanie z metod bazujących na filtracji czasoprzestrzennej. Pozwalają one wykrywać tego typu ruch, kosztem znacznego zwiększenia złożoności obliczeniowej. Więcej informacji na temat opisywanej w tym podrozdziale szybkiej transformacji falkowej można znaleźć w [18].

3 Implementacja wybranych algorytmów obliczających przepływ optyczny w języku C/C++

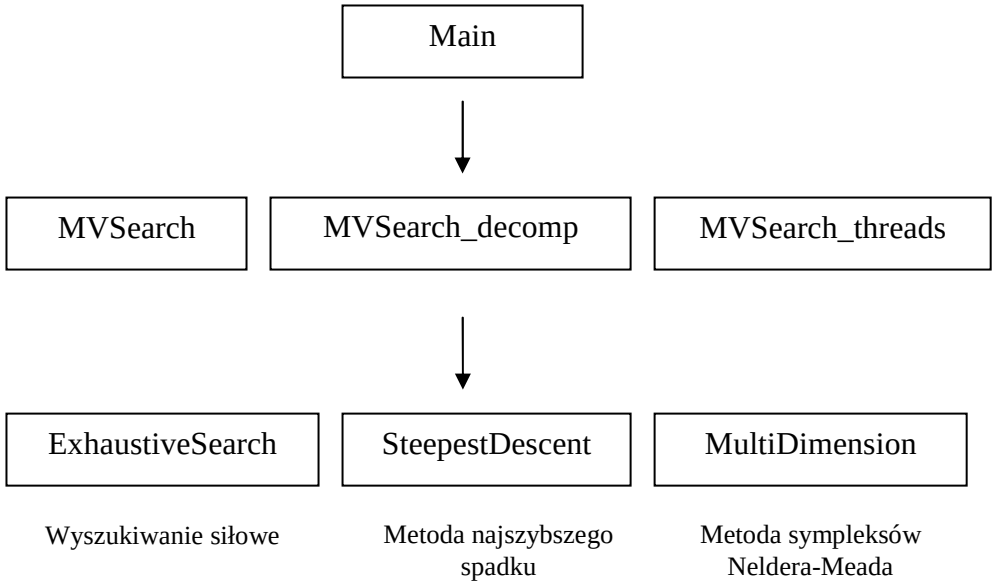
W tym rozdziale przedstawiono autorskie realizacje wyznaczania pola wektorów ruchu na podstawie omówionych wcześniej metod. Najpierw zaprezentowana zostanie implementacja interpolacji 2D, jak również różne sposoby obliczania jej jądra, następnie metoda siłowa, a w kolejnym podrozdziale algorytmy minimalizacji gradientowej i bezgradientowej. W podrozdziale 3.4 omówiona zostanie predykcyjna transformata falkowa, zastosowana w tej pracy do dekompozycji obrazów. Wobec dużej złożoności obliczeniowej omawianych procedur, wykorzystano język C++ do ich implementacji, aby przyspieszyć wykonywanie programów. W tym celu również, poza stosowaniem zoptymalizowanych wersji algorytmów, użyto specjalnych bibliotek matematycznych GSL i BLAS, zwiększających wydajność odpowiednio zaimplementowanych przekształceń. Wszystkie funkcje zostały napisane w zintegrowanym środowisku programistycznym Kdevelop pracującym w środowisku Linux. Zastosowanie systemu i narzędzi o otwartym kodzie służyło stworzeniu programów, które będą przenośne i możliwe do rozwijania w przyszłości na innych platformach sprzętowych i programowych. Ponadto praca w systemach typu Unix, umożliwia bezpłatne korzystanie z wielu wspomagających aplikacji i łatwą implementację wielowątkową, która jest konieczna ze względu na złożoność opisywanego algorytmu.

Rozdział jest uzupełniony o kilka przykładowych skryptów w języku Matlab, które służyły m.in. interpolacji obrazów testowych, generacji siatek wektorów, implementacji algorytmów w celu porównania działania z funkcjami napisanymi w języku C++, jak również weryfikacji otrzymanych wyników estymacji przepływu optycznego.

3.1 Struktura programów

Wszystkie programy zostały napisane z myślą o jak największej optymalizacji i przyspieszeniu obliczeń. Dlatego stworzone zostały typy danych w formie struktur, zawierających niezbędne dane wykorzystywane do komunikacji pomiędzy niższymi częściami (funkcjami) programu. Struktury tworzone są dynamicznie, a przesyłanie ich pomiędzy funkcjami programu odbywa się poprzez wskaźniki, aby uniknąć niepotrzebnego kopiowania danych.

Program główny, w którym uruchomiony może być algorytm wyznaczania wektorów ruchu istnieje w różnych formach: standarowej, z dekompozycją falkową, wielowątkowej. Każdy z nich zawiera pętlę obliczającą przepływ optyczny dla całego obrazka – po wszystkich wektorach. Dla każdego wektora można wybrać jedną z metod wyszukiwania: siłową, gradientową lub bezgradientową. Na rysunku 3.1 przedstawiono strukturę programu.



Rys. 3.1 Struktura programu do testowania metod wyszukiwania przepływu optycznego.

Danymi wejściowymi w algorytmie są obrazy i siatki wektorów, natomiast wyjściowymi wektory ruchu. W fazie testów wczytywanie obrazów było przeprowadzone za pomocą bibliotek VTK (ang. *Visualisation Toolkit*), umożliwiających wygodne operowanie i konwersje pomiędzy różnymi typami plików graficznych. Do reprezentacji większości danych (w tym jasności pikseli) w programie wybrano typ „float”, ponieważ ma on wystarczającą precyzję jak również można go wykorzystać podczas obliczeń z użyciem bibliotek BLAS oraz instrukcji SSE3.

W każdym z opisywanych algorytmów pojawiają się te same struktury danych i funkcje, charakterystyczne dla metod „block matching”. Są to przede wszystkim procedury interpolujące, implementowane w celu osiągnięcia pod-pikselowej dokładności, porównujące bloki, obliczające funkcje celu etc. Struktura przechowująca dane o jednym wektorze ruchu została zaprezentowana w tabeli 3-1.

Tabela 3-1 Struktura przechowująca dane o pojedynczym wektorze ruchu**.**

```
typedef struct motionVector
{
       float x;   //współrzędna X początku wektora
       float y;   //współrzędna Y początku wektora
       float dx;  //składowa X prędkości wektora ruchu
       float dy;  //składowa Y prędkości wektora ruchu
       float fit; //miara dopasowania bloku reprezentowanego przez dany wektor
}mVector;
```

3.2 Interpolacja 2D

Spośród wielu procedur służących interpolacji w tej pracy zastosowano metodę dwusześcienną. Implementacja jej jądra sprowadza się do zapisania w języku programowania odpowiednich wzorów przedstawionych w rozdziale 1.2. Kod źródłowy tej procedury został zaprezentowany w tabeli 3-2.

Tabela 3-2 Wyznaczanie współczynników jądra interpolacji.

```
void findKernelBICUBIC( float *kerCoeff, float koffsetx, float koffsety )
{
   for( int x=-1; x<=2; x++ )
      for( int y=-1; y<=2; y++ )
         kerCoeff[ (y+1)*4 + (x+1) ] = R2(abs(x-koffsetx))*R2(abs(koffsety-y));
}

/*Cubic Convolution – Robert G. Keys*/
float R2( float x )
{
     float a = -0.5;
     float x3 = x*x*x;
     float x2 = x*x;

     return (x<1) ? (a+2)*x3-(a+3)*x2+1 : ((x<2) ? a*x3-5*a*x2+8*a*x-4*a : 0.0 );
}
```

Analizy czasu wykonywania interpolacji pokazały, że powyższy kod, będący odwzorowaniem wzoru (14), nie jest najszybszym sposobem obliczania jądra interpolacji. Z tego względu przetestowane zostały liczne implementacje filtru interpolującego. Różniły się one zarówno jakością generowanych wyników jak i czasem wykonywania. Ostatecznie, autor wzorując się na kodzie źródłowym funkcji programu Matlab, stworzył procedurę interpolującą, która mimo innego schematu obliczeń dawała identyczne rezultaty. Implementacja tej procedury została zaprezentowana w tabeli 3-3.

Tabela 3-3 Szybkie generowanie współczynników jądra interpolacji.

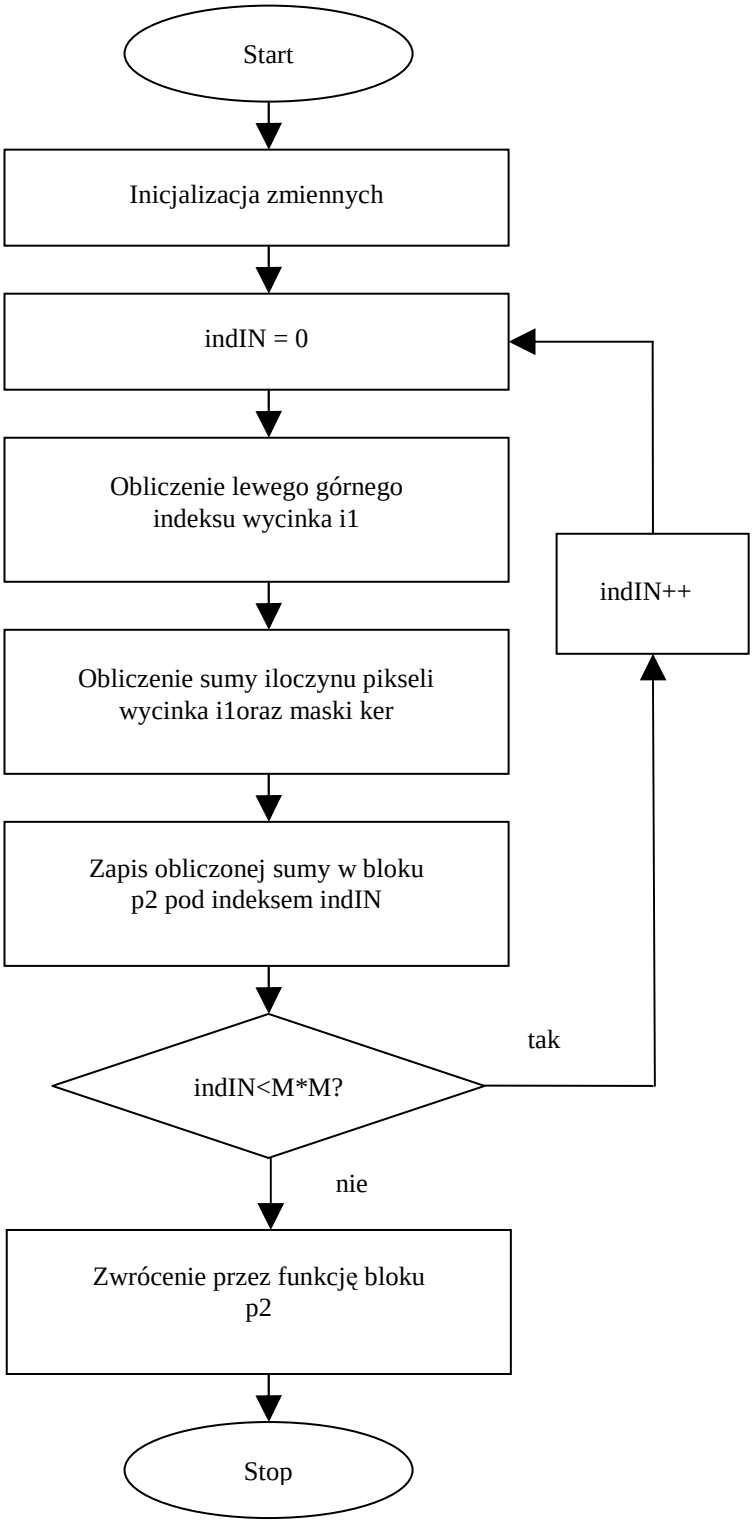
```
void findKernelBICUBIC_2( float *kerCoeff, float koffsetx, float koffsety )
{
     //polyX – funkcje obliczające określone wielomiany
     float p0x = poly0(koffsetx), p1x = poly1(koffsetx), p2x = poly2(koffsetx),
           p3x = poly3(koffsetx);
     float p0y = poly0(koffsety), p1y = poly1(koffsety), p2y = poly2(koffsety),
           p3y = poly3(koffsety);

     kerCoeff[0] = p0y*p0x*0.25; kerCoeff[1] = p0y*p1x*0.25;
     kerCoeff[2] = p0y*p2x*0.25; kerCoeff[3] = p0y*p3x*0.25;

     kerCoeff[4] = p1y*p0x*0.25; kerCoeff[5] = p1y*p1x*0.25;
     kerCoeff[6] = p1y*p2x*0.25; kerCoeff[7] = p1y*p3x*0.25;

     kerCoeff[8] = p2y*p0x*0.25; kerCoeff[9] = p2y*p1x*0.25;
     kerCoeff[10] = p2y*p2x*0.25; kerCoeff[11] = p2y*p3x*0.25;

     kerCoeff[12] = p3y*p0x*0.25; kerCoeff[13] = p3y*p1x*0.25;
     kerCoeff[14] = p3y*p2x*0.25; kerCoeff[15] = p3y*p3x*0.25;
}
```

Jądro jest obliczane zawsze dla kolejnych przesunięć wektorów. Mając wyznaczone jego współczynniki można dokonać interpolacji – wymnożenia odpowiednich pikseli bloku z maską. Ważne jest tutaj założenie, że dla uproszczenia, wszystkie piksele bloku interpolowane są o to samo przesunięcie. Algorytm interpolacji dla bloku MxM pikseli przedstawiono na rysunku 3.2. Opis zmiennych występujących na schemacie i ich odpowiedniki w kodzie programu:

- indIN – indeks kolejnych elementów interpolowanych bloków, indIN = 1…MxM,
- i1 – obraz wejściowy, imgIN->img,
- p2 – wyjściowy blok po interpolacji, imgOUT->img,
- ker – współczynniki maski interpolacji, o wielkości 4x4 piksele, kernel->kCoeff.

W omawianej pracy, do poszukiwania wektorów ruchu, stosowane są bloki 8x8 pikseli, więc liczba M na schemacie 3.2 jest równa 8. W trakcie modyfikowania programu i pracy nad optymalizacją, algorytm interpolacji również podlegał pewnym zmianom. Jedną z nich było wyciągnięcie obliczania wierzchołka wycinka obrazu mnożonego z maską interpolacji, poza obszar głównej pętli. Okazało się również, że nie ma konieczności przekazywanie do funkcji całej struktury (imgSP) z wszystkimi szukanymi wartościami

nowych pikseli. Wobec powyższego założenia, o równym przesunięciu wszystkich pikseli interpolowanego bloku, wystarczyły jedynie dwa parametry: przemieszczenie w osi X i Y.



Rys. 3.2 Schemat blokowy interpolacji

3.3 Metoda Siłowa

Metoda wyszukiwania siłowego została przedstawiona w rozdziale drugim. Umieszczony również został tam, stosowany w tej pracy algorytm znajdowania wektorów ruchu. Jest to metoda bardzo stabilna, jednak czasochłonna, pod względem obliczeniowym. Na duży czas wykonywania programu wpływa głównie wielokrotne wywoływanie funkcji interpolującej. Dlatego tak ważne było jej przyspieszenie. Przykładowo dla obszaru poszukiwań ±4.05 piksela z próbkowaniem co 0.1, wyszukanie jednego wektora wymagało 6724 interpolacji. Pewnym sposobem optymalizacji było obliczanie jądra interpolacji poza główną pętlą. Jest to możliwe ponieważ w metodzie siłowej wykorzystuje się relatywnie niewielką liczbę jąder interpolacji, np. dla rozdzielczości 0.1 piksela wymaganych jest 100 różnych jąder. Mimo to znalezienie 1155 takich wektorów, przy parametrach podanych powyżej, wymagało na komputerze klasy Pentium M 1,86Ghz ponad 100 sekund. Optymalizacja funkcji interpolującej opisana w poprzednim paragrafie, nie polepszyła znacznie otrzymywanych wyników i uwidoczniła wyraźną potrzebę zastosowania innych metod.

Poza interpolacją najważniejsze procedury charakterystyczne dla algorytmu siłowego, jak również innych metod to `desiredPatch`, `calculatePatchSP`, `calculateFit` i `findAndSetBestFit`. Druga z wymienionych funkcji, zajmuje się wyliczaniem bloku pikseli o nowych rzeczywistych położeniach. W fazie optymalizacji została wyeliminowana, ze względu na wspomniane wyżej założenie dotyczące równego przesunięcia całego interpolowanego bloku. Szczegóły dotyczące opisu omawianych funkcji znajdują się w Dodatku A.

3.4 Metody minimalizacji wieloparametrowej

Metoda gradientowa i bezgradientowa wykorzystywane do minimalizacji funkcji celu pozwoliły na otrzymanie dużo lepszych rezultatów pod względem czasu wykonywania programu. Obie zostały przedstawione w rozdziale drugim. Poniżej znajdują się informacje związane z implementacją tych algorytmów.

3.4.1 Metoda najszybszego spadku

Algorytm najszybszego spadku został zaczerpnięty z [19]. Po zastosowaniu kilku modyfikacji i napisaniu procedury obliczającej funkcję celu i wyznaczającej gradient,

możliwe było użycie go, do wyznaczania pola przepływu optycznego. Czas obliczeń jest znacznie krótszy, gdyż dla znalezienia jednego wektora jest wykonywanych maksymalnie 20 iteracji. Wliczając funkcje obliczające gradient, jest to równoważne kilkudziesięciu wywołaniom procedury interpolacji. Ich ilość jest o dwa rzędy mniejsza niż przy metodzie siłowej co obrazuje duży zysk w czasie wykonywania programu. Tolerancja została ustawiona na 0.01, oznacza to, że poniżej tej wartości, przyjmowanej przez funkcję celu, następuje przerwanie obliczeń.

3.4.2 Metoda sympleksów Neldera-Meada

Metoda bezgradientowa, zwana algorytmem sympleksów Neldera-Meada, wykorzystywana w tej pracy, pochodzi ze zbioru bibliotek GSL (ang. *GNU Scientific Library*). Charakterystyczną cechą tej metody jest brak obliczania pochodnych, co powoduje kolejny zysk czasowy. Dzięki temu uzyskuje ona najlepszą wydajność z omawianych sposobów wyznaczania przepływu optycznego. Również w tej metodzie tolerancja została ustawiona na wartość 0.01. W tabeli 3-4 przedstawiono przykładowe użycie metody sympleksowej przy korzystaniu z biblioteki GSL.

Danymi wejściowymi do funkcji obliczającej wektory ruchu metodą sympleksów są struktury:

- przechowująca dane o pojedynczym wektorze, który np. na skutek analizy wielorozdzielczej zmienia punkt swojego zaczepienia, mv,
- przechowująca dane o wektorze ruchu, mvOrg,
- struktury z wartościami jasności pikseli obrazów, img1, img2,
- struktura zawierająca informacje dotyczące typu użytej interpolacji, kernel,
- założona tolerancja, tol.

Rezultat działania funkcji:

- obliczenie składowych prędkości dx i dy wektora ruchu i zapisanie ich w zmiennej mv.

Tabela 3-4 Algorytm Neldera-Meada pochodzący ze zbioru bibliotek GSL.

```
Void MultiDimension( mVector *mv, mVector *mvOrg, const imgInput *img1, const
imgInput *img2, intKernel *kernel,float tol )
{
	sGrid *sg = new sGrid;
	sg->X = 8;   // wymiar bloku
	sg->Y = 8;

	float *patch2 = new float[ sg->X*sg->Y ];
	desiredPatch(patch2, img2, mvOrg, sg );
	img11 = img1; //przypisanie do zmiennej globalnej, uzywanej w f. celu
	size_t iter = 0;
	int status;
	double size;

	const gsl_multimin_fminimizer_type *T =gsl_multimin_fminimizer_nmsimplex;
	gsl_multimin_fminimizer *s = NULL;

	gsl_vector *ss,*x;
	gsl_multimin_function minex_func;

	x = gsl_vector_alloc(2);	//punkt startowy
	gsl_vector_set(x,0,mv->x);
	gsl_vector_set(x,1,mv->y);

	ss = gsl_vector_alloc(2);
	gsl_vector_set_all(ss, 1.0);  //krok początkowy

	minex_func.n = 2;	//wymiar zadania
	minex_func.f = &my_f;  //funkcja celu
	minex_func.params = (void *)patch2; //blok wycięty z obrazu 2.

	s = gsl_multimin_fminimizer_alloc(T,2);
	gsl_multimin_fminimizer_set(s, &minex_func, x, ss); //inicjalizacja metody

	do
	{
		iter++;
		status = gsl_multimin_fminimizer_iterate(s);
		if(status) break;

		/*sprawdzenie rozmiaru sympleksu*/
		size = gsl_multimin_fminimizer_size(s);
		status = gsl_multimin_test_size(size,tol);

	}while(status == GSL_CONTINUE && iter < 50);

	mv->dx = (float)gsl_vector_get(s->x,0) - mv->x;
	mv->dy = (float)gsl_vector_get(s->x,1) - mv->y;

	gsl_multimin_fminimizer_free(s);
	gsl_vector_free(ss);
	gsl_vector_free(x);
	delete [] patch2;
	delete sg;
}
```

3.5 Dekompozycja 2D

Dekompozycja, która w tej pracy jest narzędziem znacznie przyspieszającym, obliczenia powinna być zaimplementowana w taki sposób, aby charakteryzowała się dużą szybkością działania przy zachowaniu wysokiej jakości generowanych pod-obrazów. Precyzyjność została otrzymana dzięki zastosowaniu falek, natomiast krótki czas dekompozycji zapewnił algorytm szybkiej predykcyjnej transformaty falkowej (ang. *The Fast Lifting Wavelet Transform*). Podobnie jak w przypadku interpolacji, algorytm napisany został na podstawie zasady działania wbudowanych funkcji Matlaba, służących analizie wielorozdzielczej. Omawiana transformata składa się z kilku naprzemiennych stopni predykcji i uaktualnienia. Obie te funkcja spełnia jedna procedura: `lsupdate`, w zależności od parametrów jakie są do niej przekazywane. Do zapewnienia dobrej jakości dekompozycji wystarczyła najprostsza falka: Haara. Algorytm polegał na podziale próbek na parzyste i nieparzyste. Następnie wyjściowy sygnał detali otrzymywany był jako błąd predykcji próbek nieparzystych na podstawie parzystych. Natomiast sygnał aproksymacji uzyskano jako efekt działania operatora uaktualnienia i sumy obu grup sygnałów. W tabeli 3-5 przedstawiono fragment kodu dokonujący podziału sygnału wejściowego i wyliczający dekompozycję w jednym wymiarze – wzdłuż wierszy. Następnie, otrzymane wyniki macierzy aproksymacji i detali, ponownie są dzielone na parzyste i nieparzyste elementy oraz dokonywana jest filtracja, tym razem wzdłuż kolejnych kolumn obu pod-obrazów. Efektem jest wtedy dekompozycja całego obrazu. Danymi wejściowymi funkcji zajmującej się analizą wielorozdzielczą są:

- struktura zawierająca jasności pikseli dekompowanego obrazu,
- wagi filtrów P i U w predykcyjnej transformacji falkowej.

Rezultatem jest zestaw czterech macierzy współczynników falkowych zawierających łącznie tyle elementów, co obraz wejściowy. Nadpisują one go w taki sposób, że część będąca aproksymacją jest zawsze umieszczona w jego lewym górnym rogu.

Tabela 3-5 Fragment kodu dekompozycji.

```
/*Podział na próbki parzyste i nie parzyste*/
for(i=firstIdxAPP; t_ind=i<<1, i < sLH; i++)
{
	L[i]=img->img[ t_ind ];
	H[i]=img->img[ t_ind + 1];
}

int NBL = wCoeff->Y;
float *y1, *y2;
float *liftFilt;

/*Filtracja typu predykcja-uaktualnienie*/
for(i = 0; i<NBL; i++)
{
	ind = i*(wCoeff->X);
	liftFilt = wCoeff->coeff + (ind+2);
	switch((int)wCoeff->coeff[ind])
	{
	case -1:
		y1 = lsupdate(1,H,liftFilt,(int)wCoeff->coeff[ind+1],
		   x_sLH,y_sLH);
		for( k=0; k<sLH; k++ )
		     L[k] += (y1[k]);
		delete [] y1;
		break;
	case -2:
		y1 = lsupdate(1,L,liftFilt,(int)wCoeff->coeff[ind+1],
		   x_sLH,y_sLH);
		for( k=0; k<sLH; k++ )
		     H[k] += (y1[k]);
		delete [] y1;
		break;
	}
}
```

3.6 Zastosowanie bibliotek GSL i BLAS

W celu zwiększenia szybkości zaimplementowanych metod, warto również skorzystać z optymalizacji, poprzez zastosowanie zoptymalizowanych pod kątem konkretnego procesora niektórych operacji matematycznych. W projekcie zostały wykorzystane dostępne bezpłatnie biblioteki GSL (ang. *GNU Scientific Library*) i BLAS (ang. Basic Linear Algebra Subprograms). Pierwsze z nich są zbiorem napisanych w języku C, matematycznych funkcji służących wielu różnym numerycznym operacjom. W pakiecie GSL można znaleźć algorytmy sortujące, operacje na macierzach i wektorach, transformacji Fouriera, metody minimalizacji i wiele innych. Są w nim również dostępne wspomniane powyżej biblioteki BLAS w wersji wysoko i nisko poziomowej. Biblioteka BLAS stanowi zbiór wysokiej jakości procedur numerycznych służących do przeprowadzania operacji algebraicznych na wektorach i macierzach. Wersja wysoko-poziomowa dostępna w GSL jest kompatybilna ze strukturami jakimi operują inne funkcje

tego zbioru, przez co jednak jest mniej wydajna. Natomiast nisko-poziomowa pozwala na większe przyspieszenie obliczeń. Mimo to najlepszą wydajność zapewniają biblioteki BLAS optymalizowane pod konkretną architekturę. Najbardziej znane implementacje to ACML (ang. *AMD Core Math Library*) i MKL (ang. *Math Kernel Library*) będący produktem Intela. W tej pracy autor korzystał głównie z tego drugiego zbioru. Porównanie działania bibliotek „zwykłych" i dedykowanych pod określony sprzęt znajduje się w tym rozdziale.

W prezentowanej pracy, spośród różnych narzędzi dostarczonych przez GSL, zostały wykorzystane funkcje służące minimalizacji wielowymiarowej. Z kilku testowanych metod, w pełni udało się zastosować algorytm simpleksowy Neldera-Meada, który mimo, że nie daje możliwości ingerencji w jego wewnętrzną strukturę lecz jedynie wywoływany jest przez określone procedury, osiąga najlepszą szybkość i dokładność spośród testowanych algorytmów.

Stosowanie bibliotek BLAS i ich użyteczność nie jest tak oczywista jak mogłoby się wydawać. Mimo używania wersji adekwatnych do procesora, nie zawsze można osiągnąć lepsze wyniki niż stosując standardowe operacje. Dzieje się tak, gdyż operacje BLAS nie są wydajne przy obliczeniach na wektorach lub macierzach zawierających niewielką liczbę elementów. W takim przypadku wywołanie funkcji realizującej operacje matematyczne zajmuje więcej czasu niż same obliczenia. Ponadto na uwagę zasługuje ich zróżnicowanie ze względu na funkcjonalność. Istnieją następujące poziomy:

- poziom 1 – operacje wektor-wektor,
- poziom 2 – operacje macierz-wektor,
- poziom 3 – operacje macierz-macierz.

W zależności od zastosowania powinno korzystać się z funkcji charakterystycznych dla danego poziomu.

3.6.1 Linkowanie

Istotny dla stosowania bibliotek BLAS jest sposób ich linkowania z projektem. Szczególnie jest to ważne jeżeli równocześnie korzysta się z pakietu GSL, tak jak było w przypadku tego projektu. W opisywanej pracy były używane zoptymalizowane biblioteki Intela ze zbioru MKL. Jeżeli jednak były linkowane w drugiej kolejności (po bibliotekach GSL), to programy korzystały z wersji nie zoptymalizowanych, mimo że nie było włączonych ich plików nagłówkowych i nie były ustawione odpowiednie flagi. Jest to najprawdopodobniej spowodowane faktem implementacji niezoptymalizowanych bibliotek

BLAS bezpośrednio w GSL. W celu otrzymania wydajności charakterystycznej dla bibliotek zoptymalizowanych należy, więc pamiętać o linkowaniu pakietu MKL w pierwszej kolejności.

3.6.2 Test szybkości BLAS

Praktyczne testy przeprowadzone w fazie pisania programów zostały zaprezentowane w tym podrozdziale. Sprawdzony został głównie fakt, czy biblioteki BLAS dla odpowiednio dużej ilości danych są bardziej wydajne. Ponadto przedstawione jest porównanie pomiędzy biblioteką BLAS „zwykłą" i zoptymalizowaną pod konkretny procesor, a także algorytmem wykorzystującym standardowe biblioteki matematyczne w języku C/C++.

Funkcja `cblas_sdot` umożliwia mnożenie wektora przez wektor, a następnie sumowanie wyników mnożeń. Dane, na których były testowane szybkości wykonywania, to dwa wektory typu float. Jeden z nich wypełniony był danymi o wartościach 1.0, drugi natomiast 2.5. Otrzymane wyniki są zaprezentowane w tabeli 3-6.

Tabela 3-6 Test szybkości BLAS na dwóch testowych wektorach

Ilość elementów w wektorze	Czas [s] dla 10^8 iteracji		
	Operacje standardowe	BLAS	BLAS zoptymalizowany
25	5.56	5.84	9.21
50	10.22	10.60	10.90
75	14.89	15.24	12.27
100	19.37	20.62	13.65
1000	188.2	184.0	68.2
2000	370.6	367.8	130.2

3.6.3 Wykorzystanie biblioteki BLAS w algorytmach obliczających przepływ optyczny

Z analizy tabeli 3-6 wynika, że efektywne wykorzystanie biblioteki BLAS wymaga odpowiednio skonstruowanego algorytmu. W omawianych metodach obliczania przepływu optycznego podjęto próby przyspieszenie programów przez zastosowanie bibliotek BLAS,

jednak było to możliwe tylko w części zaimplementowanych funkcji. Największe przyspieszenie otrzymano dla dekompozycji obrazów. Zastosowano tam wielokrotnie funkcję pierwszego poziomu `cblas_saxpy`. Efektem wykorzystania zoptymalizowanej wersji BLAS było trzydziestoprocentowe przyspieszenie procesu dekompozycji obrazów. Kluczem do sukcesu były operacje na odpowiednio dużych wektorach. Funkcja ta została wywołana 2000 razy, a rezultaty zaprezentowano w tabeli 3-7.

Tabela 3-7 Porównanie czasów dekompozycji dla obliczeń z wykorzystaniem biblioteki BLAS oraz standardowej biblioteki matematycznej języka C/C++.

Dekompozycja Oryginalna	Dekompozycja z wykorzystaniem BLAS
20.09[s]	13.44[s]

Podjęto również próbę przyspieszenia interpolacji. W tym celu prztestowano funkcje biblioteki BLAS ze wszystkich trzech poziomów. Zastosowanie poziomów 2 (macierz-wektor) i 3 (macierz-macierz), mogło być konkurencyjne ze względu na charakter interpolacji: mnożenie wycinka obrazu z maską, a następnie sumowanie pomnożonych elementów – operacja charakterystyczne dla macierzy. Zastosowano następujące funkcje dla poszczególnych poziomów biblioteki BLAS:

- `cblas_sdot` – pierwszy poziom (wektor-wektor),
- `cblas_sgemv` – drugi poziom (macierz-wektor),
- `cblas_sgemm`. – trzeci poziom (macierz-macierz).

Uzyskane rezultaty dowiodły, że interpolacji małych bloków o wymiarach 8x8, nie można przyspieszyć za pomocą opisywanych bibliotek. Ilość danych w mnożonych buforach jest zbyt mała, a ich przekształcenie do odpowiednio dużych struktur, trwa zbyt długo, aby algorytmy mogły działać w sposób efektywny. Czasy wykonywania dla jednej pary obrazów przy wykorzystaniu różnych procedur interpolacji przedstawiono w tabeli 3-8. Przedstawione tam oznaczenia funkcji wyjaśniono poniżej:

- FAST2 – najlepsza wersja interpolacji, wykorzystuje standardową bibliotekę C/C++,
- CBLAS (wektor-wektor) – interpolacja z zastosowaniem BLAS pierwszego poziomu,

- CBLAS2 (wektor-wektor) – interpolacja z zastosowaniem BLAS pierwszego poziomu, zmieniona struktura programu, aby możliwy był większy zysk z zastosowania bibliotek,
- CBLAS3 (macierz-wektor) – interpolacja z zastosowaniem BLAS drugiego poziomu, szybkość najlepsza spośród funkcji korzystających z BLAS,
- CBLAS3 (macierz-macierz) – interpolacja z zastosowaniem BLAS trzeciego poziomu, najmniej efektywna czasowo.

Tabela 3-8 Porównanie czasów osiąganych przez algorytm przy zastosowaniu różnych funkcji interpolujących

FAST2	CBLAS (w-w)	CBLAS2 (w-w)	CBLAS3 (m-w)	CBLAS3 (m-m)
0.21[s]	0.38[s]	0.33[s]	0.29[s]	0.67[s]

Szczegółowy opis zaprezentowanych procedur znajduje się w dodatku A. Z otrzymanych czasów wykonywania można wywnioskować, że najlepszą wydajnością charakteryzuje się funkcja korzystająca z bibliotek standardowych.

3.7 Zastosowanie wielowątkowości

Ze względu na strukturę algorytmu, duży zysk można osiągnąć zrównoleglając algorytm poprzez wykorzystanie wielu rdzeni procesora w tym samym czasie. Autor ze względu na posiadany procesor jednordzeniowy, nie sprawdził zależności czasowych dla programów wielowątkowych, stworzył jednak funkcję, która może być wykorzystywana na innych komputerach do zrównoleglenia wykonywanych obliczeń. Przykład implementacji dla używanych metod obliczania przepływu optycznego znajduje się w tabeli 3-9. Przedstawiono tam podejście polegające na podzieleniu fragmentów obrazka pomiędzy stworzone wątki, które wykonują niezbędne obliczenia równocześnie na wielu jednostkach obliczeniowych. W przytoczonym kodzie źródłowym ukazano jedynie sposób ich generowania i funkcję wywoływaną przez wykreowany proces. Na uwagę zasługuje tu sposób przekazania danych do wątku. Polega on na przesłaniu ich, za pomocą wskaźnika rzutowanego na typ void, do funkcji która jest wywoływana przez stworzony proces. Aby móc przekazać dużą ilość danych w ten sposób, konieczne jest zgromadzenie ich w

strukturze, do której wskaźnik należy przesłać do opisywanej procedury. W prezentowanym kodzie źródłowym danymi wejściowymi podobnie jak w poprzednich przypadkach są obrazy, współrzędne początków wektorów ruchu, tolerancja, a ponadto zakres określający, które wektory są przetwarzane przez dany rdzeń. Natomiast wielkościami wyjściowymi są ponownie wartości prędkości dx i dy wektorów ruchu, scalane w dobrej kolejności dzięki funkcji `pthread_join` mającej na celu oczekiwanie na zakończenie poprzedniego wątku.

Tabela 3-9 Deklaracja wątku i funkcji przez niego wywoływanej

```
Pthread_t thread1, thread2, thread3, thread4; //deklaracja wątków
int  iret1, iret2, iret3, iret4;

pThrd *data1 = new pThrd;  //zbiór danych używany w jednym z wątków
data1->mvs = mvs;
data1->img1 = img1;        //obraz pierwszy
data1->img2 = img2;        //obraz drugi
data1->kernel = kernel;
data1->tol = 0.01f;        //zadana tolerancja
data1->start = 0;   //zakres wektorów
data1->stop = 300;  //przetwarzanych w jednym watku

/*stworzenie wątku*/

iret1 = pthread_create( &thread1, NULL, Pthread_fun, (void*)data1);

(…)

/*funkcja wywoływana przez wątek*/

void *Pthread_fun( void *ptr )
{
       pThrd *data = new pThrd;
       data = (pThrd*)ptr;
       mVector *mvs = data->mvs;
       const imgInput *img1 = data->img1;
       const imgInput *img2 = data->img2;
       intKernel *kernel = data->kernel;
       float tol = data->tol;
       int start = data->start;
       int stop = data->stop;

       for(int i = start; i<=stop; i++)    //metoda symleksowa
              MultiDimension( &mvs[i], &mvs[i], img1, img2, kernel, tol  );
}
```

3.8 Weryfikacja wybranych etapów zadania w języku Matlab

Pierwszą częścią weryfikacji było analizowanie zaimplementowanej w języku C interpolacji i jej podobieństwa do analogicznej funkcji Matlaba, traktowanej jako wzorzec dokładności. To właśnie o wyborze ostatecznej metody wykorzystywanej przy obliczaniu przepływu optycznego decydowały testy wyników interpolacji. W tym przypadku porównywano dwa obrazy, z czego jeden był efektem znajdywania nowych wartości

pikseli przez funkcję napisaną w języku C, a drugi – wynikiem działania procedury Matlaba. Porównanie wykonano za pomocą obliczania średniej bezwzględnych różnic wartości pikseli obu obrazów.

Korzystając z języka Matlab wygenerowano materiał testowy w postaci obrazów (interpolowanych) i wzorcowych pól wektorów. Ponadto w omawianym programie tworzono również siatki wektorów ruchu, które zapisane do plików były następnie odczytywane przez program implementowany w języku C++.

Algorytmy wyznaczające pole wektorów ruchu, testowane były głównie na syntetycznych obrazach wygenerowanych według sposobu przedstawionego w rozdziale 1.2. Dzięki temu znane były przesunięcia o jakie poddano interpolacji obraz pierwszy. Dlatego obliczając przepływ optyczny, można było go porównać z polem wektorów wzorcowych. Na podstawie pierwszego obrazu sprawdzano czy przepływ optyczny stworzony w procesie interpolacji i wyznaczony przez algorytm pokrywają się i zachowują ogólną tendencję co do kierunku. Natomiast drugi wykres umożliwiał znacznie pełniejszą analizę, obrazując nawet najmniejsze różnice pomiędzy pojedynczymi wektorami.

Kolejnym testem było zaimplementowanie algorytmu siłowego w języku Matlab i porównanie otrzymywanych wyników z pochodzącymi z programu napisanego w C++. W obu przypadkach wykorzystano te same dwa obrazki do analizy. Rezultaty wykazały zbieżność obu implementacji. Różnice w średnich błędach estymacji wektorów, pojawiały się dopiero na czwartym miejscu po przecinku. Był to kolejny argument, potwierdzający poprawność implementacji procedury interpolacji w języku C/C++. Również w celu sprawdzenia zaimplementowany został algorytm gradientowy najszybszego spadku, który potwierdził z dużą precyzją, rezultaty otrzymane z programu napisanego w środowisku kDevelop.

Ponadto funkcje służące dwuwymiarowej falkowej dekompozycji obrazów, także wzorowane były na swoich odpowiednikach z Matlaba. Dodatkowo cennym narzędziem okazał się on przy przygotowaniu i rozkładzie obrazów testowanych na poszczególne składowe barwne w przestrzeniach RGB czy YUV.

4 Analizy wyników pomiarów

Czwarty rozdział przedstawia szczegółowe porównanie testowanych algorytmów zarówno pod względem dokładności jak i szybkości. Każda metoda oceniona została poprzez pomiar trzech parametrów: błędu średniego, błędu maksymalnego, czasu wykonywania algorytmu dla dwóch obrazów. Przedstawione zostały problemy jakie mogą wyniknąć ze stosowania poszczególnych strategii, jak również sposoby radzenia sobie z nimi. Zaprezentowano analizy obrazów syntetycznych i rzeczywistych. Do tej pory opisywane rozważania dotyczące obliczania przepływu optycznego opierały się na obrazach monochromatycznych. W tym rozdziale zostaną przedstawione testy na obrazkach kolorowych, gdyż z takimi mamy do czynienia w sekwencjami video większości aplikacji systemów wizyjnych. Ukazano również wpływ poszczególnych składowych barwnych na osiągane rezultaty estymacji pola wektorów ruchu.

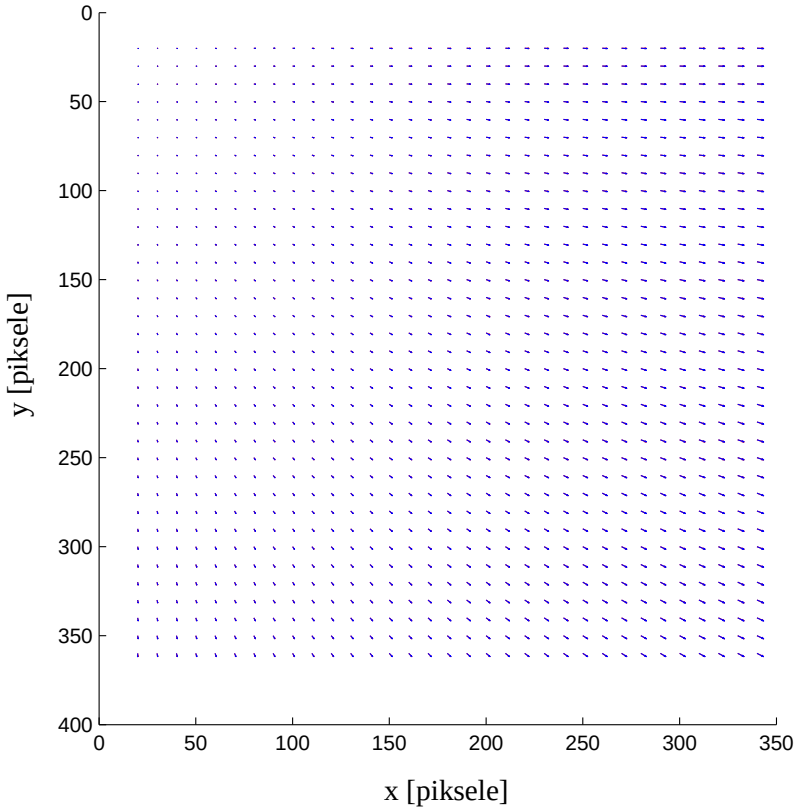
4.1 Weryfikacja dokładności i czasu wykonywania zaimplementowanych metod

Zaprezentowane w tej pracy algorytmy w pierwszej kolejności testowane były na obrazach syntetycznych, aby możliwe było wyznaczenie dokładności, poprzez porównywanie z posiadanym wzorcem. Parę obrazów sprawdzono dla wersji algorytmów bez i używających dekompozycji, aby zbadać jej wpływ na jakość i szybkość estymacji.

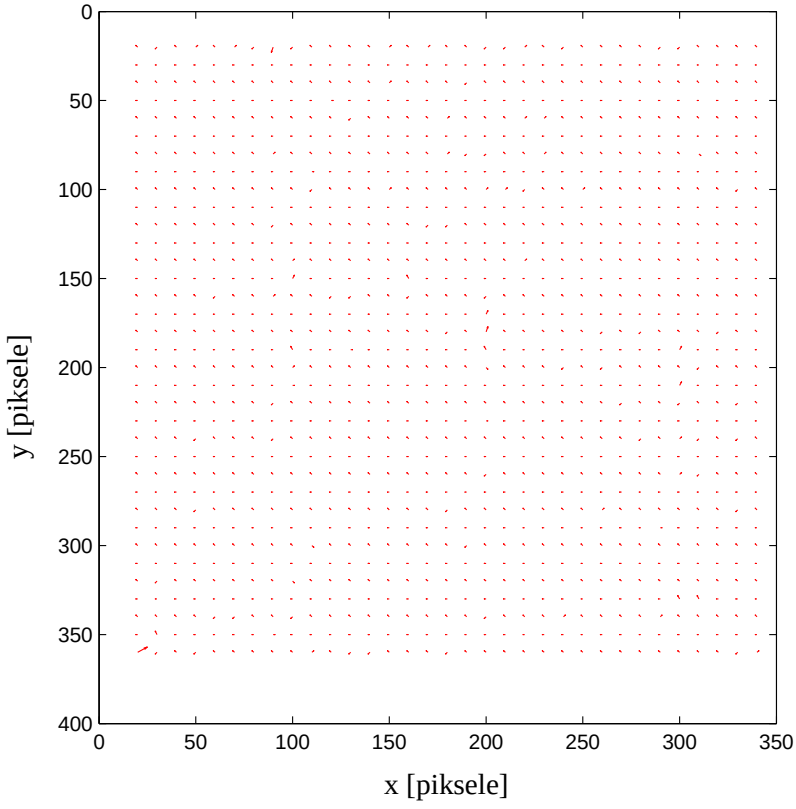
Jak już wspomniano wcześniej, „sztuczny” obraz otrzymano wskutek rozciągnięcia obrazu rzeczywistego. Sposób i wybór materiału do analizy opisano już w podrozdziale 1.5, natomiast weryfikację wyników estymacji, wykonywano w programie Matlab.

4.1.1 Metoda siłowa

Dla prawidłowo dobranego obszaru poszukiwań znalezienie poprawnych wektorów ruchu przedstawianą metodą cechuje duże prawdopodobieństwo w porównaniu do innych omawianych metod. Drugą ważną wielkością jest rozdzielczość, standardowo dobierana w badanych problemach, na wartość 0.1. Efekty poprawnego ustawienia parametrów algorytmu prezentują rysunki 4.1 i 4.3.



Rys. 4.1 Wektory ruchu obliczone (kolor niebieski) nałożone na wektory ruchu wzorcowe



Rys. 4.2 Dwudziestokrotnie powiększone różnice (błąd) długości wektorów wzorcowych i obliczonych.

Na rysunku 4.3 przedstawiono jakość wyznaczania przepływu optycznego, w przypadku gdy dobrany jest odpowiednio duży obszar poszukiwań. Dodatkowym wskaźnikiem dokładności są wartości błędu średniego, każdorazowo obliczane na podstawie danych do wykresu różnic wektorów:

$$\overline{E} = \frac{\sum_{i=1}^{K} \sqrt{\left(u_w(i) - u(i)\right)^2 + \left(v_w(i) - v(i)\right)^2}}{K} \tag{24}$$

gdzie: u_w – wektor przesunięć wzorcowych wzdłuż osi X,

v_w – wektor przesunięć wzorcowych wzdłuż osi Y,

u – wektor przesunięć obliczonych wzdłuż osi X,

v – wektor przesunięć obliczonych wzdłuż osi Y,

K – ilość iteracji równa 1155.

Dla każdej metody, liczony jest również błąd maksymalny:

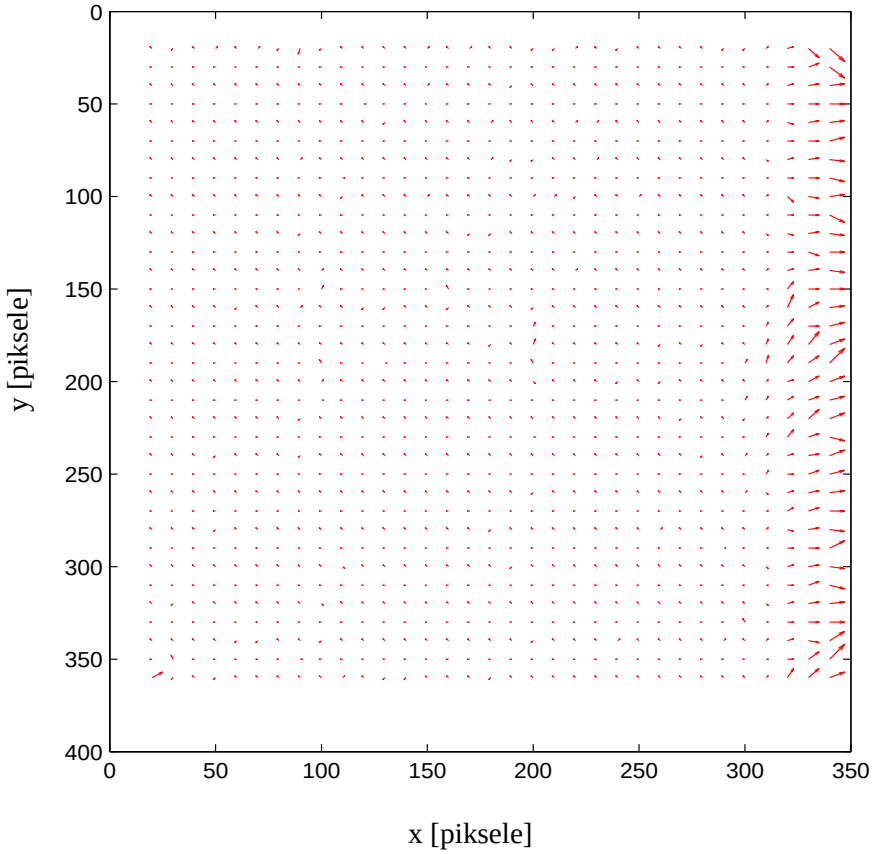
$$E^{\max} = \max\left(\sqrt{\left(u_w(i) - u(i)\right)^2 + \left(v_w(i) - v(i)\right)^2}\right), \quad i = 1...1155 \tag{25}$$

Natomiast wyznacznikiem szybkości jest czas wykonywania algorytmu bez uwzględniania czasu wczytywania obrazów za pomocą VTK. Wyniki dla prezentowanej metody przedstawia tabela 4.1.

Tabel 4-1 Wyniki błędów i czasu obliczeń dla metody siłowej z rozdzielczością 0.1.

Blad średni	0,0615 [piksela]
Błąd maks.	0,2915 [piksela]
Czas	111 [s]

W metodzie siłowej ważne jest dobranie odpowiedniego obszaru poszukiwań, który wynosi ±4.05 pikseli dla wyników zaprezentowanych na rysunkach 4.1 i 4.3. Można zaobserwować, że podany zakres był wystarczający, aby dokonać dobrej estymacji wektorów ruchu. Przypadek, który obrazuje wybranie zbyt małego obszaru poszukiwań przedstawiono na wykresie 4.4.



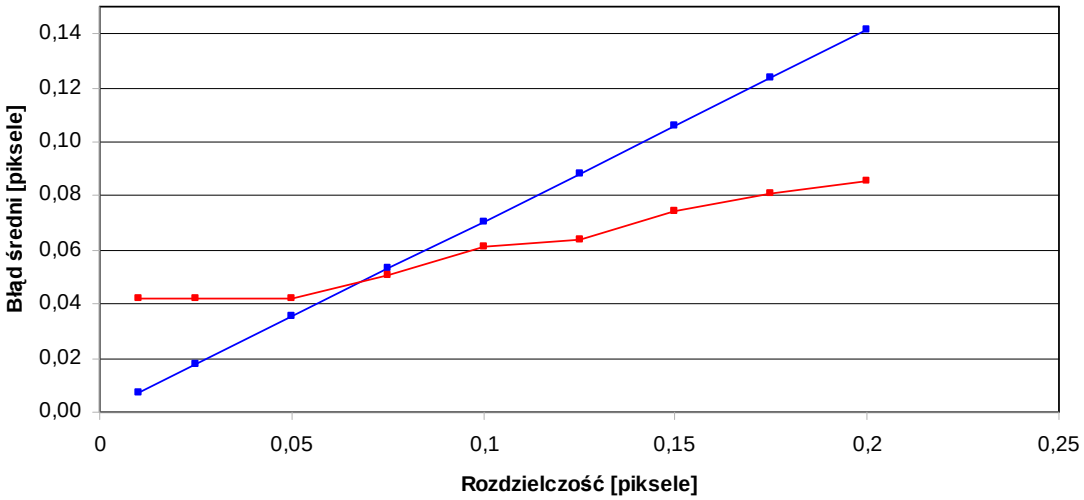
Rys. 4.3 Dwudziestokrotne różnice długości wektorów wzorcowych i obliczonych. Skutki zastosowania zbyt małego obszaru poszukiwań.

Występowanie błędów o znacznie większej wartości przy prawej krawędzi obrazu zostało spowodowane tym, że wzorcowe wektory w tej części rysunku są dłuższe niż maksymalne przesunięcia wnoszone przez zakres obszaru poszukiwań.

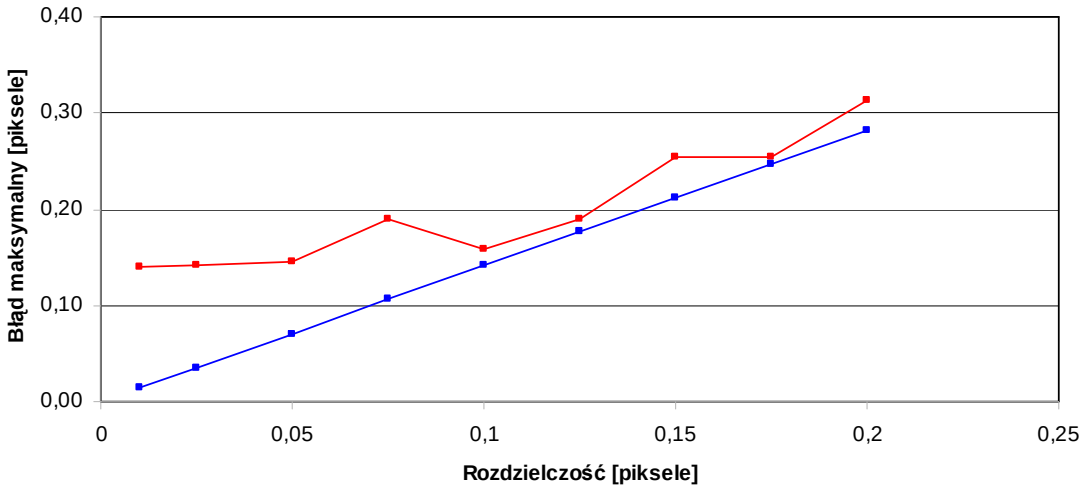
Kolejnym ważnym czynnikiem w metodzie siłowej jest rozdzielczość z jaką wyznaczane są wektory ruchu. Jej wartość określa co jaką ułamkową część piksela dokonuje się wyszukiwanie nowych przemieszczeń i interpolacja. W celu przebadania wpływu przyjmowanej w algorytmie rozdzielczości, przeprowadzono kilka symulacji dla takich samych parametrów lecz zmieniającej się jej wartości. Otrzymane wyniki przedstawione, są na rysunkach 4.4 i 4.5. Na podstawie uzyskanych wykresów można wstępnie oszacować jaką dokładność jest się w stanie uzyskać w obliczaniu przepływu optycznego na testowanych obrazach. Daje to cenną informację, na podstawie której można weryfikować precyzję osiąganą, później badanymi, metodami zoptymalizowanymi. Wartości teoretyczne, występujące na wykresach, zostały wyznaczone w zależności od obranej rozdzielczości.

$$E_t^{\max} = r\sqrt{2} \tag{26}$$

$$\overline{E}_t = r\frac{\sqrt{2}}{2} \tag{27}$$



Rys. 4.4 Błąd średni estymacji wektorów ruchu w funkcji rozdzielczości dla metody siłowej, kolor niebieski – wartości teoretyczne, kolor czerwony – wartości obliczone.



Rys. 4.5 Błąd maksymalny estymacji wektorów ruchu w funkcji rozdzielczości dla metody siłowej, kolor niebieski – wartości teoretyczne, kolor czerwony – wartości obliczone.

Na uwagę zasługuje rysunek 4.4, na którym można zaobserwować, że dla rozdzielczości większej niż 0.1 wartości obliczone charakteryzują się mniejszym błędem niż ich teoretyczne odpowiedniki. Może być to spowodowane tym, że wartości pikseli jasności w testowym obrazie otrzymane zostały poprzez dokonywanie interpolacji o setne części

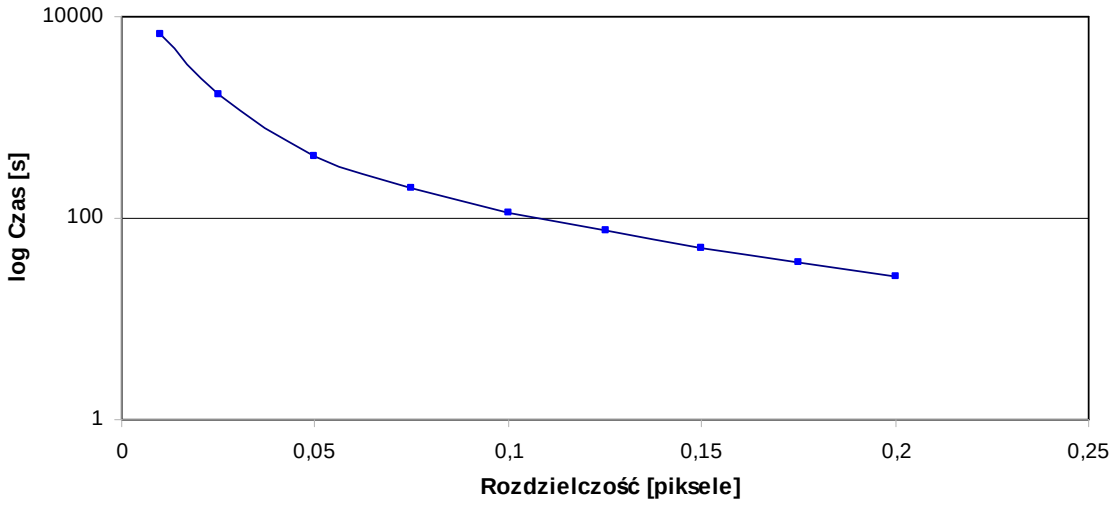
piksela, którego wielokrotnościami są 0.1, 0.15 i 0.2. Ze zmniejszającą się rozdzielczością widoczny jest wyraźny spadek wartości teoretycznej w stosunku do obliczonej. Oznacza to skończoną ilość informacji, jakie mogą występować w obrazie, zgodnie z tzw. pojemnością Shannona. Należy zwrócić również uwagę, że od rozdzielczości równej 0.05, obliczona dokładność poprawia się nieznacznie. Jednakże, aby porównywać następne metody z najlepszą dokładnością jaką udało się uzyskać metodą siłową przedstawiono w tabeli 4-2 wyniki otrzymane dla r = 0.01.

Tabela 4-2 Wartości błędów i czasu obliczeń dla metody siłowej z rozdzielczością 0.01.

Blad średni	0,0420 [piksela]
Błąd maks.	0,1401 [piksela]
Czas	6800 [s]

Bardzo długi czas wykonywania algorytmu, zarówno dla rozdzielczości równej 0.1 (111s) i 0.01 (6800s), wskazuje konieczność stosowania innych metod i obrazuje stopień trudności problemu. Znakomicie przedstawia to wykres czasu wykonywania programu jako funkcji zmieniającej się rozdzielczości., przedstawiony na rysunku 4.6.

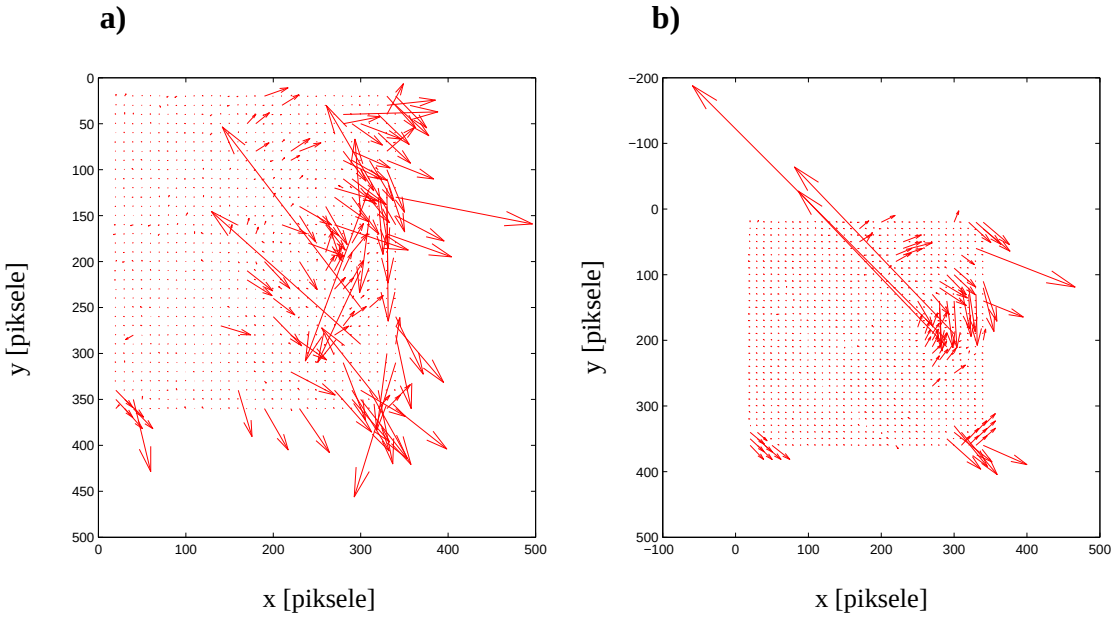
Mimo usprawnień interpolacji, omówionych w podrozdziale 3.2, najlepszy czas jaki udało się osiągnąć tą metodą to 21.92 s dla r=0.1 i 1300 s dla r=0.01. Prezentowane szybkości wykonywania kolejnych algorytmów liczono dla najszybszej wersji interpolatora.



Rys. 4.6 Wykres zależności czasu wykonywania algorytmu siłowego od rozdzielczości z jaką wyszukiwane są nowe wektory.

4.1.2 Metoda Najszybszego Spadku

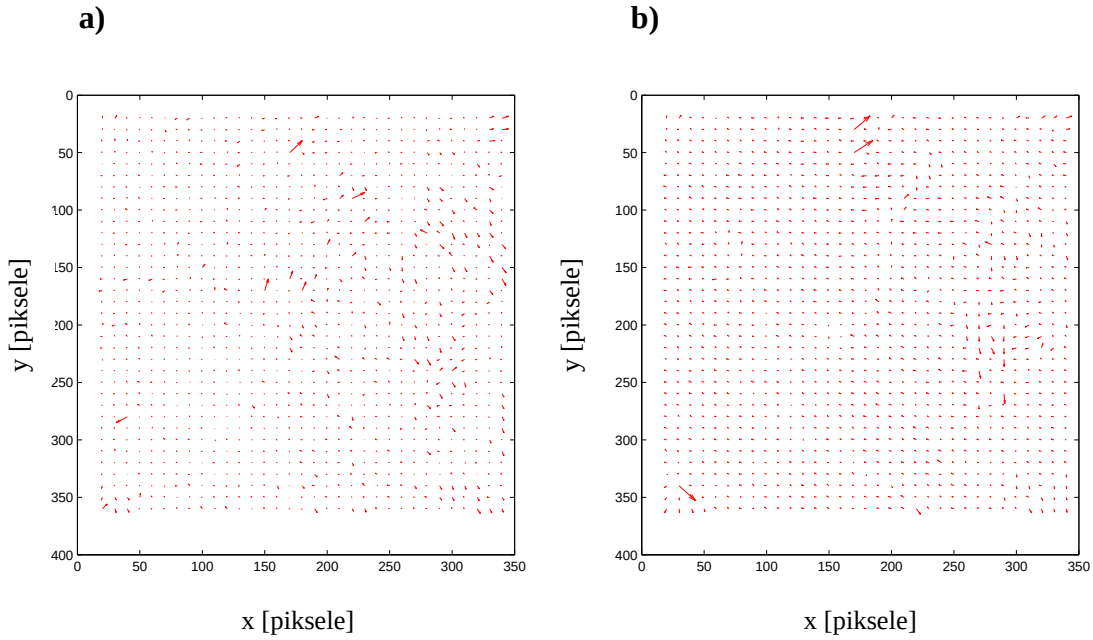
Dla algorytmów minimalizujących funkcję celu, posiadających mniejszą złożoność obliczeniową niż przeszukiwanie siłowe, tolerancja została ustawiona na 0.01, aby zapewnić referencję do metody siłowej z rozdzielczością równą 0.01. Oznacza to, że położenia wektorów będą osiągane, przy wartości funkcji celu mniejszej niż założony powyżej parametr. Metody minimalizacji najszybszego spadku wrażliwe są jednak na minima lokalne i często przez to popełniają błędy w wyznaczaniu wektorów. Przykładem obrazu różnic wektorów dla tej metody jest rysunek 4.7.



Rys. 4.7 Dwudziestokrotnie powiększone różnice długości wektorów wzorcowych i obliczonych metodą najszybszego spadku: a) zastosowanie bloków 8x8, b) zastosowanie bloków 16x16.

Widoczne są liczne grube błędy w wyznaczaniu wektorów ruchu. Świadczy to niewątpliwie o istnieniu minimów lokalnych, szczególnie w obszarze gdzie przesunięcia między obrazami są większe. Jednym ze sposobów pozbycia się błędów jest zastosowanie większego rozmiaru porównywanych bloków. W tym celu zmieniono ich wymiary z 8x8 na 16x16. Na rysunku 4.7b) można zaobserwować, że wiele małych błędów zostało zniwelowanych, jednak zwiększenie rozmiaru poddawanych komparacji fragmentów obrazów, powoduje ryzyko błędnego ich skorelowania. Dzieje się tak, ponieważ bloki obejmują duże obszary, których kawałki w rzeczywistości przemieszczają się w różny sposób. Wyraża się to w kilku dużych błędach w centralnej części rysunku. O ile pomyłek

na rogach, związanych z efektem krawędzi, można nie brać pod uwagę, to pozostałe są niedopuszczalne. Z tego względu należy zaproponować inne rozwiązanie. Dlatego spróbowano obliczenia mediany z otoczenia każdego z obliczonych wektorów (w przypadku wektorów położonych blisko krawędzi jest to mediana z bezpośrednio sąsiadującej z nim grupy wektorów). Następnie jeżeli przynajmniej jedna ze składowych przemieszczenia wektora, różni się od wartości wyznaczonej mediany o minimum 0,5 piksela jest przez nią zastępowana. Dało to pozytywne wyniki, które zostały zaprezentowane na rysunku 4.8.



Rys. 4.8 Dwudziestokrotnie powiększone różnice wektorów wzorcowych i obliczonych metodą najszybszego spadku: a) zastosowanie mediany dla bloków 8x8 pikseli, b) zastosowanie mediany dla bloków 16x16 pikseli.

Można łatwo zauważyć, że dla bloków o rozmiarach 8x8 pikseli, widać drobne błędy, które zmniejszają się dla bloków o mniejszych wymiarach. Wyliczenie błędów wskazuje jednak, że estymacja jest dokładniejsza przy komparacji mniejszych bloków. Zastosowanie grup 16x16 pikseli, mimo, że niweluje liczne nieprawidłowości, szczególnie związane z szumem w obrazie, to jednak złe skorelowanie powoduje wzrost średniej wartości błędu. Ponadto aplikacja większych bloków zwiększa znacznie czas obliczeń i czyni to podejście znacznie mniej praktycznym. Otrzymane wyniki przedstawia tabela 4-3.

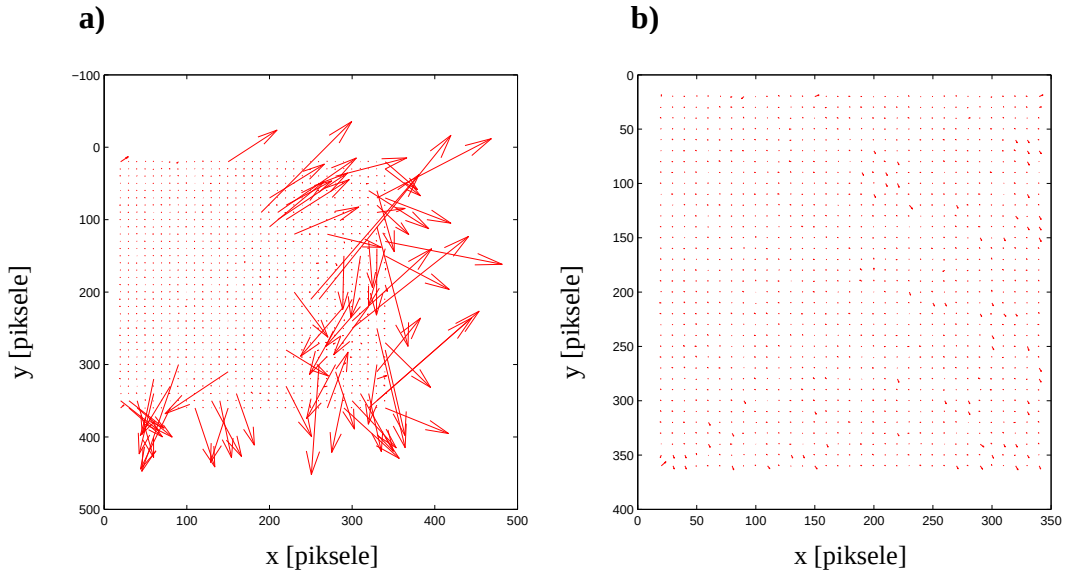
Tabela 4-3 Wartości błędów i czasu obliczeń dla metody najszybszego spadku przy założonej tolerancji 0.01.

	Bloki 8x8 pikseli	Bloki 16x16 pikseli
Blad średni [piksele]	0.0671	0.0961
Błąd maks. [piksele]	0.7210	0.9336
Czas [s]	0.69	2.04

Na podstawie otrzymanych wyników można jasno stwierdzić, że metoda gradientowa, poza mniejszą złożonością obliczeniową, umożliwia uzyskanie dokładności bliskiej algorytmowi siłowemu z r=0.1. Wyniki generowane w porównaniu do przeszukiwania siłowego z r=0.01 są znacznie gorsze: błąd średni zwiększył się o 50%, a maksymalny ok. 5 razy. Niezaprzeczalną zaletą algorytmu jest jednak dużo większa szybkość obliczeń.

4.1.3 Metoda sympleksów Neldera-Meada

Metoda Neldera-Meada, podobnie jak algorytm najszybszego spadku, znajduje często minima lokalne, zamiast globalnych. Widoczne jest to na rysunku 4.9a), który bardzo przypomina wykresy dla metody analizowanej w podrozdziale 4.1.2. Można zauważyć, że małe błędy spowodowane szumem w obrazie, prawie nie występują. Z tego powodu od razu zastosowano zastępowanie wektorów błędnych przez medianę z ich otoczenia (patrz rysunek 4.9b).



Rys. 4.9 Dwudziestokrotnie powiększone różnice pomiędzy wektorami wzorcowymi

i obliczonymi metodą sympleksów: a) efekt działania algorytmu, b) rezultat po zastosowaniu mediany.

Tabela 4-4 Wartości błędów i czasu obliczeń dla metody sympleksów przy założonej tolerancji 0.01.

Blad średni [piksele]	0.0487
Błąd maks. [piksele]	0.2762
Czas [s]	0.19

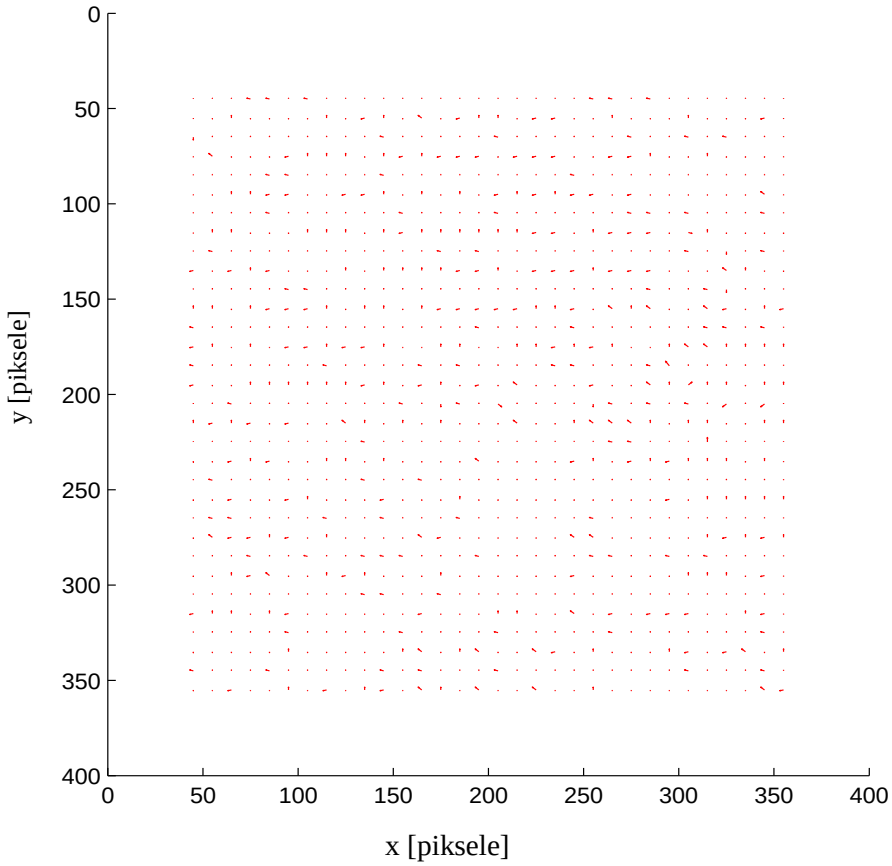
Zaprezentowana metoda, daje bardzo dobre efekty co do dokładności obliczeń. Wartość błędu średniego jest jedynie o kilka tysięcznych większa niż w przypadku algorytmu siłowego z r=0.01, natomiast błąd maksymalny jest dwukrotnie większy. Wyniki ukazują, że metoda ta nie tylko przyspiesza obliczenia, ale również polepsza jakość generowanych wektorów. Dzieje się tak, ponieważ algorytm sympleksowy, jak i gradientowy nie jest ograniczony przez określoną rozdzielczość. Ponadto metoda Neldera-Meada wykazuje większą stabilność niż podejście gradientowe Należy zwrócić uwagę na dużą prędkość wykonywania obliczeń, która jest blisko czterokrotnie szybsza niż algorytm gradientowy, a stukrotnie niż przeszukiwanie siłowe. Wszystkie opisane powyżej rezultaty przedstawiono w tabeli 4-5:

Tabela 4-5 Wartości błędów i czasu obliczeń dla opisanych metod nie wykorzystujących dekompozycji.

Stosowane algorytmy	Siłowy		Najszybszego spadku		Sympleksów Neldera-Meada
Rozdzielczość/tolerancja	0.1	0.01	0.01	0.01	0.01
Wielkość bloku [piksele]	8x8	8x8	8x8	16x16	8x8
Blad średni [piksele]	0,0615	0,0420	0.0671	0.0961	0.0487
Błąd maks. [piksele]	0,2915	0,1401	0.7210	0.9336	0.2762
Czas [s]	111	6800	0.69	2.04	0.19

4.1.4 Zastosowanie dekompozycji obrazów

Wykorzystanie hierarchicznej analizy obrazów jest narzędziem znacznie przyspieszającym i polepszającym jakość znajdowanych wektorów. Jako pierwszą wypróbowano metodę siłową. W celu przedstawienia poprawy jakości i szybkości estymacji wektorów ruchu jakie daje dekompozycja, zastosowano ją do algorytmu siłowego działającego z rozdzielczością 0.1. Ukazała ona możliwość znacznego zmniejszenia obszaru wyszukiwania, dzięki obliczeniom na najwyższy poziomie, a następnie dokonując jedynie korekt położenia wektora, przy każdorazowym przechodzeniu o poziom niżej. Ponadto na uwagę zasługuje fakt, że jeżeli przeszukiwano pewien obraz w zakresie ±4.05 piksela, to na trzecim poziomie, wystarczy jedynie sprawdzić go w przedziale ±0.55 piksela. Wykres różnic wektorów przedstawiono na rysunku 4.10.



Rys. 4.10 Dwudziestokrotnie powiększone różnice wektorów ruchu. Metoda siłowa z zastosowaniem dekompozycji.

Zastosowany jest trochę inny rozmiar obrazów i inna liczba poszukiwanych wektorów, związana z charakterystyką powstawania siatki pola przepływu optycznego, przedstawioną na rysunku 1.11. Wyniki obliczeń dla omawianego przypadku znajdują się poniżej. Wskazują one na większą dokładność metody używającej dekompozycji. Należy również zwrócić uwagę na błąd maksymalny, który zmniejszył się blisko o połowę w porównaniu do tego samego algorytmu, ale nie wykorzystującego dekompozycji. Ponadto uzyskane wartości błędów różnią się jedynie o tysięczne części piksela w porównaniu do metody z r=0.01. Szybkość wykonywania programu również zmieniła się znacząco. Spowodowane to jest głównie zmniejszeniem obszaru poszukiwań. Dodatkowo stwierdzono, że dekompozycja spowodowała poprawę dokładności algorytmu o rozdzielczości 0.1, praktycznie zrównując jego rezultaty z metodą wykorzystującą r=0.01 i skracając czas obliczeń ponad dziesięć tysięcy razy. Wartości błędów i czasu przedstawiono w tabeli 4-6.

Tabela 4-6 Wartości błędów i czasu obliczeń dla metody siłowej przy zastosowaniu dekompozycji z r=0. 1.

Blad średni [piksele]	0.0501
Błąd maks. [piksele]	0.1601
Czas [s]	0.59

Dekompozycja również pozytywnie wpłynęła na algorytm gradientowy. Dzięki wyszukiwaniu wielopoziomowemu, znacznie ograniczono popełnianie błędów, przez co można było zrezygnować ze stosowanie mediany w celu wykluczenia grubych pomyłek. Parametry użyteczności algorytmu zaprezentowano w tabeli 4-7.

Tabela 4-7 Wartości błędów i czasu obliczeń dla metody najszybszego spadku przy zastosowaniu dekompozycji.

Blad średni [piksele]	0.0531
Błąd maks. [piksele]	0.2379
Czas [s]	0.46

Wyniki potwierdzają znaczną poprawę wartości błędu średniego i maksymalnego, mimo to są one większe niż dla metody siłowej. Różnicę wektorów wzorcowych i wyznaczony zaprezentowano na rysunku 4.11a). Blisko dwukrotnie w porównaniu do analogicznego

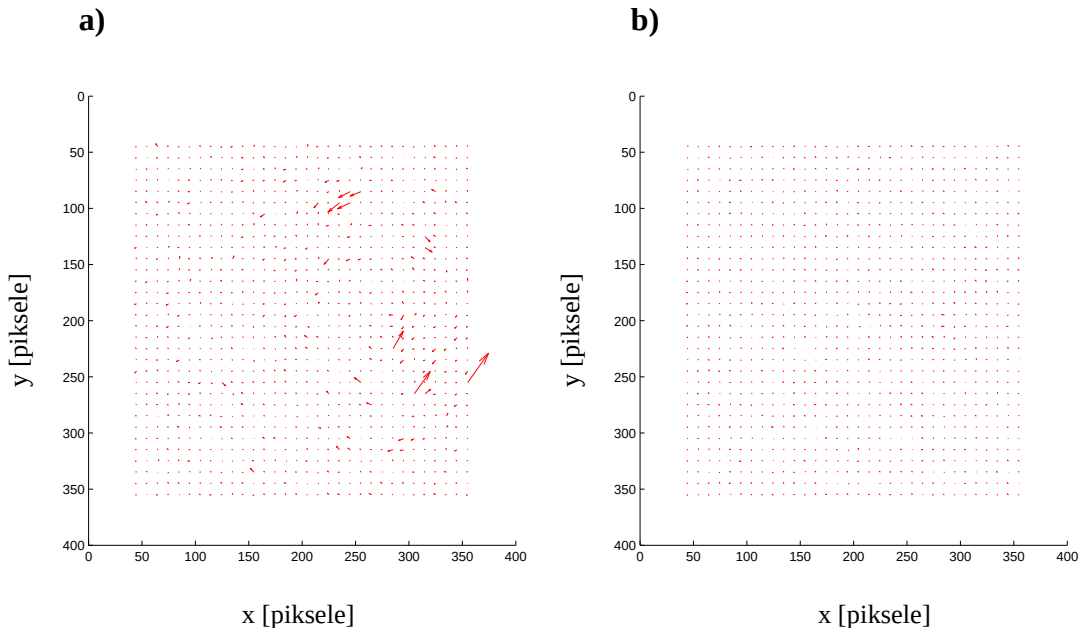
algorytmu bez dekompozycji, skrócił się czas obliczeń, jednak zysk nie był tak znaczący jak w przypadku metody siłowej.

Najbardziej pozytywne efekty wykazało zastosowanie dekompozycji w algorytmie sympleksowym (rysunek 4.11b). Nie wystąpiły żadne grube błędy w wyszukiwaniu wektorów ruchu, a osiągnięta dokładność, jest zbliżona do najlepszej jakości, jaka może być osiągnięta (patrz rysunki 4.4 i 4.5). Wyniki prezentuje tabela 4-8.

Tabela 4-8 Wartości błędów i czasu obliczeń dla metody sympleksowej przy zastosowaniu dekompozycji.

Blad średni [piksele]	0.0442
Błąd maks. [piksele]	0.1323
Czas [s]	0.16

Co ważne, pomiary błędów zostały wyznaczone bez stosowania mediany do źle wyliczonych wektorów, z czego wynika, że nie ma obszaru, gdzie algorytm traciłby stabilność.



Rys. 4.11 Dwudziestokrotnie powiększone różnice długości wektorów ruchu: a) metoda gradientowa z dekompozycją, b)metoda sympleksowa z dekompozycją.

Na uwagę zasługuje fakt, że błąd średni metody sympleksowej jest niemal identyczny jak dla metody siłowej z rozdzielczością równą 0.01, a błąd maksymalny nawet mniejszy o

0.1. Ponadto również czas jest najkrótszy ze wszystkich rozważanych metod. Wszystkie wyniki dla algorytmów stosujących dekompozycję zaprezentowano w tabeli 4-9.

Tabela 4-9 Wartości błędów i czasu obliczeń dla opisanych metod wykorzystujących dekompozycję.

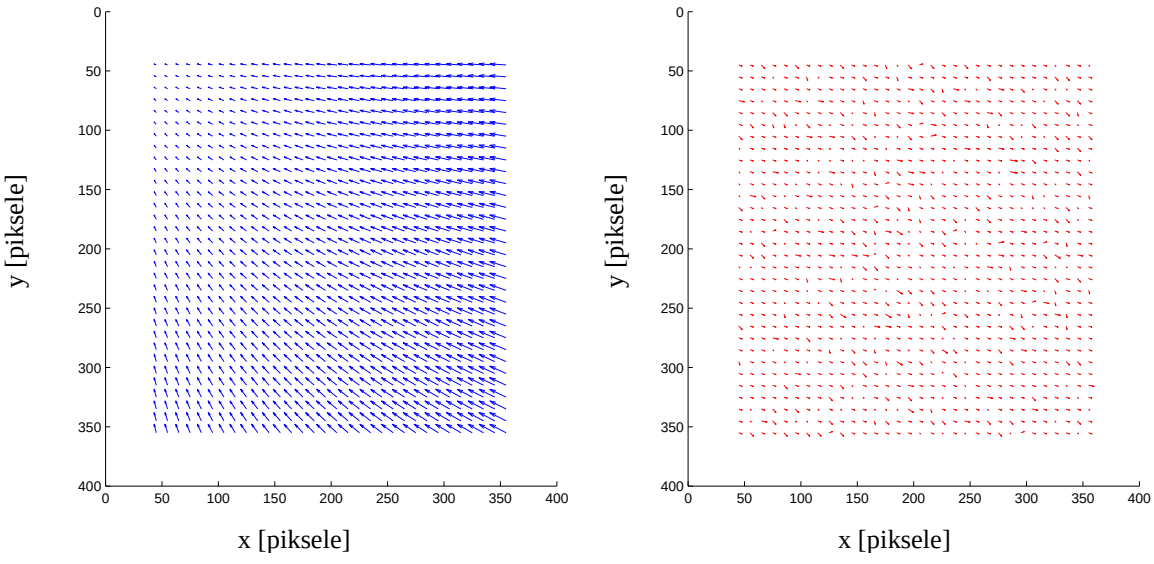
Stosowane algorytmy	Siłowy	Najszybszego spadku	Sympleksów Neldera-Meada
Blad średni [piksele]	0.0501	0.0531	0.0442
Błąd maks. [piksele]	0.1601	0.2379	0.1323
Czas [s]	0.59	0.46	0.16

Aby sprawdzić dokładniej pozytywny wpływ dekompozycji, jako obrazy testowe użyto:

- obrazu 1. – tego samego co dotychczas,
- obrazu 2. – interpolowanego co 1.04 w osi X i co 1.02 w osi Y.

Stosując metodę siłową, należy zauważyć, iż konieczne staje się zdefiniowanie większego obszaru poszukiwań, wskutek większych przesunięć wprowadzonych przy prawej krawędzi obrazu. Co jednak charakterystyczne – wystarczy zwiększyć obszar poszukiwań na najwyższym poziomie, gdyż odpowiedzialny jest on za znalezienie poprawnego fragmentu obrazu. Korekta na niższych poziomach, wpływająca na polepszenie rezultatów nie wymaga analizy dużej powierzchni. Rezultaty poszukiwań zostały przedstawione na rysunku 4.12.

a) **b)**



Rys. 4.12 Analiza poprawności metody siłowej z dekompozycją dla większych maksymalnych przesunięć na obrazie: a) wektory ruchu, b) dwudziestokrotne różnice wektorów wzorcowych i obliczonych.

Obszar poszukiwań dobrany w tym przypadku, na poziomie najwyższym wynosił ±2 piksele, natomiast na niższych poziomach i obrazach oryginalnych: ±0.5 piksela co jest wartością zbliżoną do używanej poprzednio. Jest to potwierdzeniem użyteczności zastosowanej dekompozycji. Otrzymane wyniki zaprezentowano w tabeli 4-10.

Tabela 4-10 Wartości błędów i czasu obliczeń dla metody siłowej przy zastosowaniu dekompozycji. Działanie na obrazach o dużych przesunięciach.

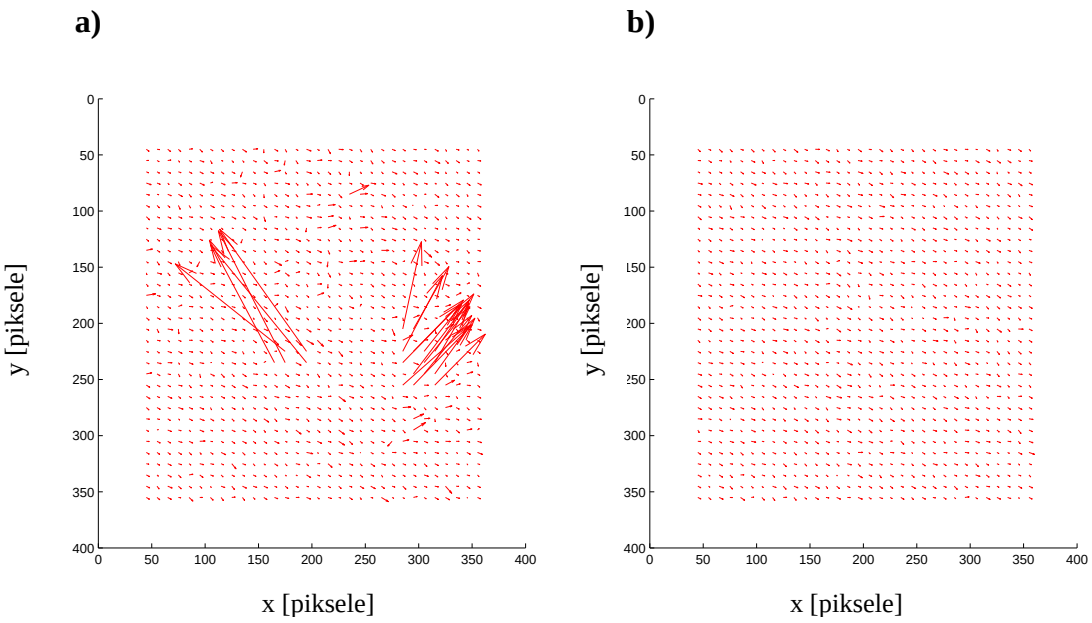
Blad średni [piksele]	0.1593
Błąd maks. [piksele]	0.3536
Czas [s]	0.72

Wyniki wskazują na wzrost średniego błędu, jednak wobec dużych przesunięć występujących między obrazami (maksymalnie ok.15 pikseli), otrzymane wyniki okazują się być dokładnymi. Interesujące rezultaty i pełniejszy obraz efektywności stosowanych metod, może dać przebadanie algorytmów zoptymalizowanych.

W metodzie gradientowej pojawiło się kilka grubych błędów co przedstawia rysunek 4.13a. Ostatecznie jednak po zastąpieniu medianą największych rozbieżności, otrzymano bardziej satysfakcjonujące wyniki, przedstawione w tabeli 4-11.

Tabela 4-11 Wartości błędów i czasu obliczeń dla metody gradientowej przy zastosowaniu dekompozycji. Działanie na obrazach o dużych przesunięciach.

Blad średni [piksele]	0.1594
Błąd maks. [piksele]	0.4401
Czas [s]	0.57



Rys. 4.13 Dwudziestokrotnie powiększone różnice pomiędzy wektorami wzorcowymi i obliczonymi: a) metoda gradientowa, b) metoda sympleksowa.

Mimo to przedstawione rezultaty są najgorsze pod względem dokładności spośród omawianych metod. Ponownie potwierdziło się, że algorytm jest szybszy niż wyszukiwanie siłowe.

Wspomniane problemy nie wystąpiły w przypadku algorytmu sympleksowego, który bez dodatkowych korekcji zapewnił dokładną estymację (patrz rysunek 4.13b). Otrzymane rezultaty przedstawiono w tabeli 4-12.

Tabela 4-12 Wartości błędów i czasu obliczeń dla metody sympleksowej przy zastosowaniu dekompozycji. Działanie na obrazach o dużych przesunięciach.

Blad średni [piksele]	0.1577
Błąd maks. [piksele]	0.2743
Czas [s]	0.17

Przedstawione rezultaty potwierdziły wcześniejsze wyniki. Najlepsza pod względem szybkości i precyzji wyszukiwania wektorów jest metoda sympleksowa. Metoda gradientowa, wprawdzie, druga w kolejności co do prędkości, nie pozwala uzyskać bardzo dokładnych rezultatów. Algorytm siłowy, mimo, że w porównaniu ze swoją pierwotną

wersją został znacząco przyspieszony, to wciąż jest najwolniejszym z przedstawionych. Cechuje go jednak duża dokładność, dzięki której można jeszcze bardziej docenić metodę sympleksów, która dorównuje algorytmowi siłowemu w kwestii precyzji, a znacznie przewyższa go pod względem czasu..

4.2 Testy na obrazach rzeczywistych

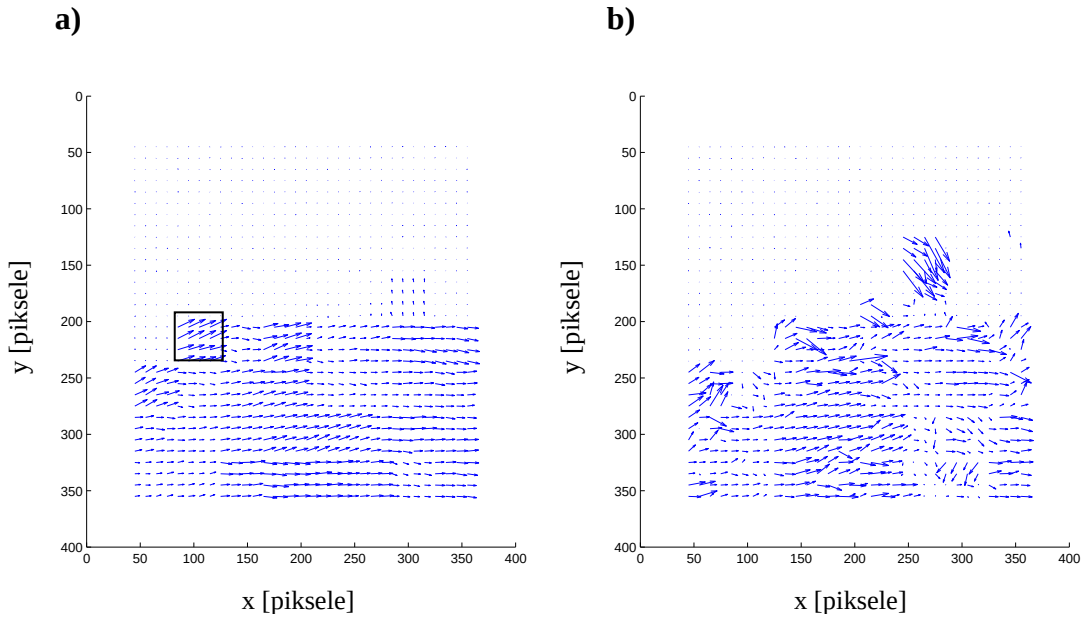
Zaprezentowane wcześniej algorytmy zastosowano również do obrazów rzeczywistych. W praktycznych zastosowaniach bowiem, algorytmy będą musiały radzić sobie z różnymi przesunięciami badanych obrazów. W opisanych w tym podrozdziale testach, nie możliwe było precyzyjne obliczenie jakości estymacji, ale jedynie ocena czy przy przesuwaniu przedmiotu wektory wskazują kierunek przemieszczającego się obiektu, czy nie występują duże rozbieżności pomiędzy nimi. Materiał testowy został wykonany aparatem cyfrowym niskiej klasy, mimo, to wyniki algorytmów pozwoliły powierzchownie ocenić jakość estymacji przepływu optycznego. Zastosowano schemat nieruchomej kamery i poruszającego się obiektu. W odstępie krótkiego czasu zostały wykonane dwa zdjęcia tego samego przedmiotu, ale nieznacznie przesuniętego.

Pierwszym poddanym testom przykładem są dwa zdjęcia myszki do komputera. Badane obrazy zostały zaprezentowane na rysunku 4.14.

a) **b)**

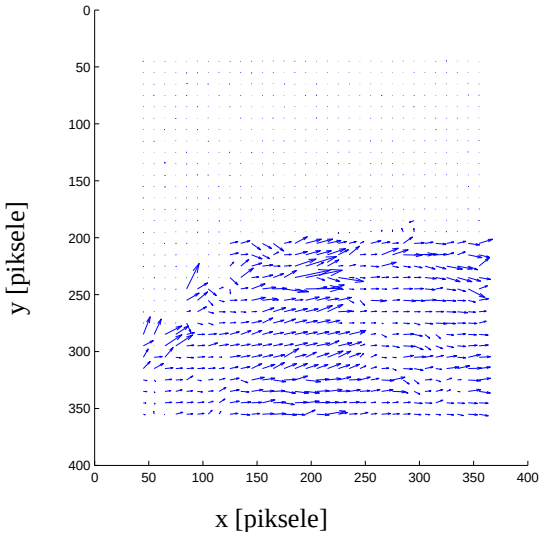
Rys. 4.14 Testowane rzeczywiste obrazy myszki do komputera

Przy analizie przepływu optycznego skorzystano z algorytmów stosujących dekompozycję obrazów, gdyż gwarantuje ona uzyskanie najlepszej dokładności. Na rysunku 4.15 Przedstawiono wyniki otrzymane dla metody siłowej i gradientowej.



Rys. 4.15 Pole wektorów ruchu wyznaczone dla rzeczywistych obrazów: a) metoda siłowa, b) metoda gradientowa.

Można zaobserwować tendencję charakterystyczną dla wcześniejszych obliczeń. Wszystkie algorytm wykazały pewną kierunkowość wyznaczonego przepływu. W metodzie siłowej zwracają uwagę grupy pikseli, przy lewej krawędzi, zwrócone w innym kierunku niż pozostałe. Są one wynikiem błędu powstałego w procesie dekompozycji. Gorsze wyniki estymacji prezentuje metoda gradientowa (patrz rysunek 4.15b), ponieważ można zaobserwować wektory z pewnością policzone błędnie. Algorytm sympleksowy, także przetworzył obrazy 4.14, w wyniku czego wygenerował następujące pole wektorów:



Rys. 4.16 Pole wektorów ruchu wyznaczone dla rzeczywistych obrazów. Metoda sympleksowa

Metoda Neldera-Meada również wykazała dobre rezultaty. Wyniki otrzymane na tych obrazach zależą jednak od wielu czynników i trudno jest dokładnie oceniać efektywność metod. Przykładowo równomierny przepływ występujący dla metody siłowej może być efektem ograniczonej siatki poszukiwań, a błędy widoczne na algorytmie sympleksowym pochodzić mogą od licznych czynników wpływających na estymację. Problemy jakie mogą wystąpić przy testowaniu obrazów rzeczywistych to:

- brak jednolitego oświetlenia,
- zniekształcenia geometryczne wywoływane przez obiektyw,
- inne rozpraszanie światła na obiekcie po przesunięciu go,
- deformacja obrazu wywołana kompresją.

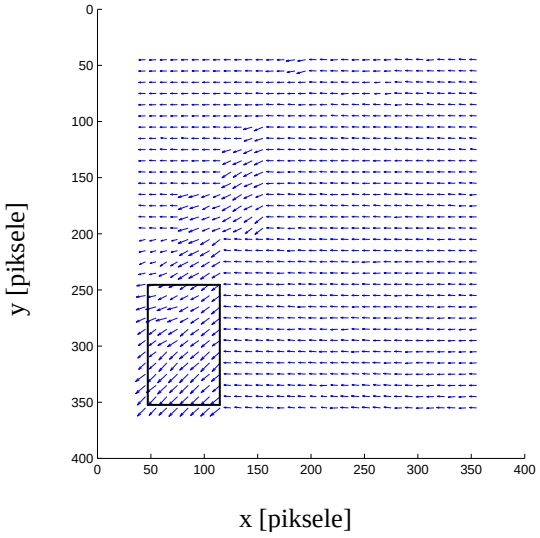
Potwierdzeniem wyżej opisanych problemów jest kolejny test, w którym także uwidaczniają się różne niekorzystne czynniki. W drugim przykładzie, przebadane zostały obrazy będące fragmentami bolidu Formuły 1 (patrz rysunek 4.17).

a) **b)**

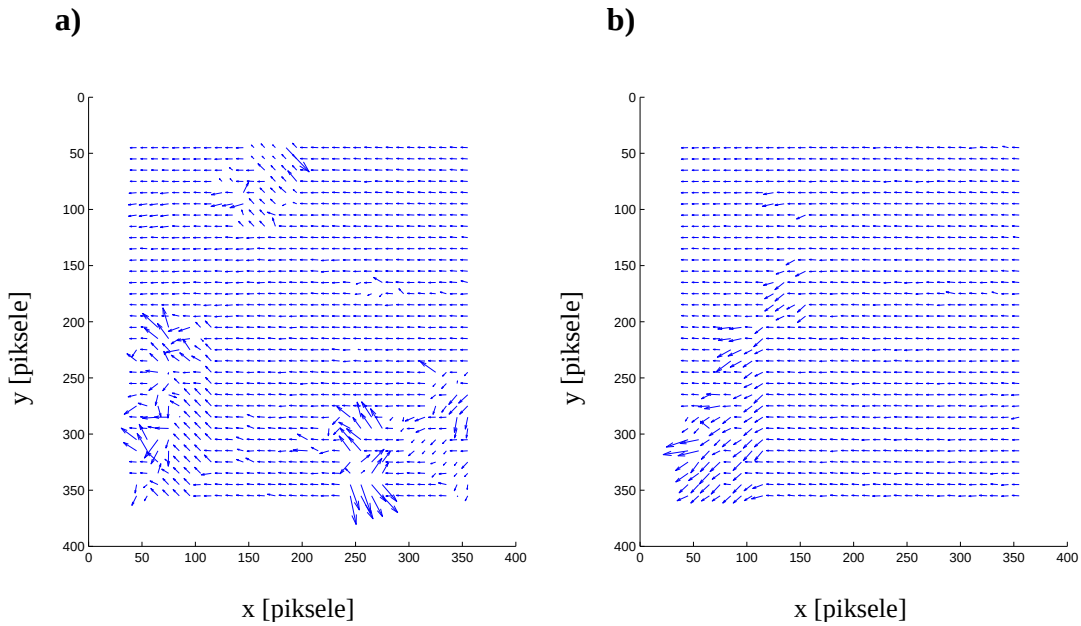
Rys. 4.17 Zdjęcia kartki przedstawiające fragment bolidu Formuły 1.

Dążono w nim do uzyskania przemieszczenia wszystkich wektorów ruchu wskutek przesunięcia całej fotografowanej ramki. W wyniku analizy pola wektorów ruchu algorytm siłowy wygenerował wyniki przedstawione na rysunku 4.18. Można na nim zauważyć, że obraz wykazuje przesunięcie w lewą stronę i większość wektorów jest ustawionych w kierunku przesuwania ramki, jedynie lewy dolny róg obrazu wykazuje inne przesunięcie.

Najwięcej błędów ponownie popełnił algorytm gradientowy, którego rezultaty zostały przedstawione na rysunku 4.19a). Można na nim zaobserwować, że w co najmniej trzech miejscach metoda ta wyliczyła wektory błędnie. Natomiast algorytm sympleksowy wykazał ponownie najlepszą jakość generowanych obliczeń (patrz rysunek 4.19b). Mimo to również w lewym dolnym rogu występują wektory o innych kierunkach niż pozostałe. Porównując te wyniki z metodą siłową, można zaobserwować, że jest ich dużo mniej. Jest to rezultatem sztywnej siatki poszukiwań metody siłowej. Warto wspomnieć, że omawiane niedokładności mogły być spowodowane zagięciem kartki, która była fotografowana lub po prostu dużą ilością jednolitych powierzchni występujących na obrazie.

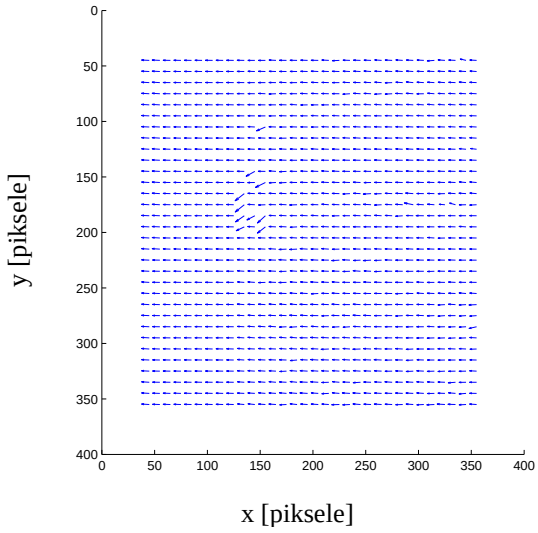


Rys. 4.18 Pole wektorów ruchu wyznaczone metodą siłową dla rzeczywistych obrazów. Zaznaczony kwadrat wynikający z błędu występującego w dekompozycji



Rys. 4.19 Pole wektorów ruchu wyznaczone obrazów fragmentu bolidu: a) metoda najszybszego spadku, b) metoda Neldera-Meada.

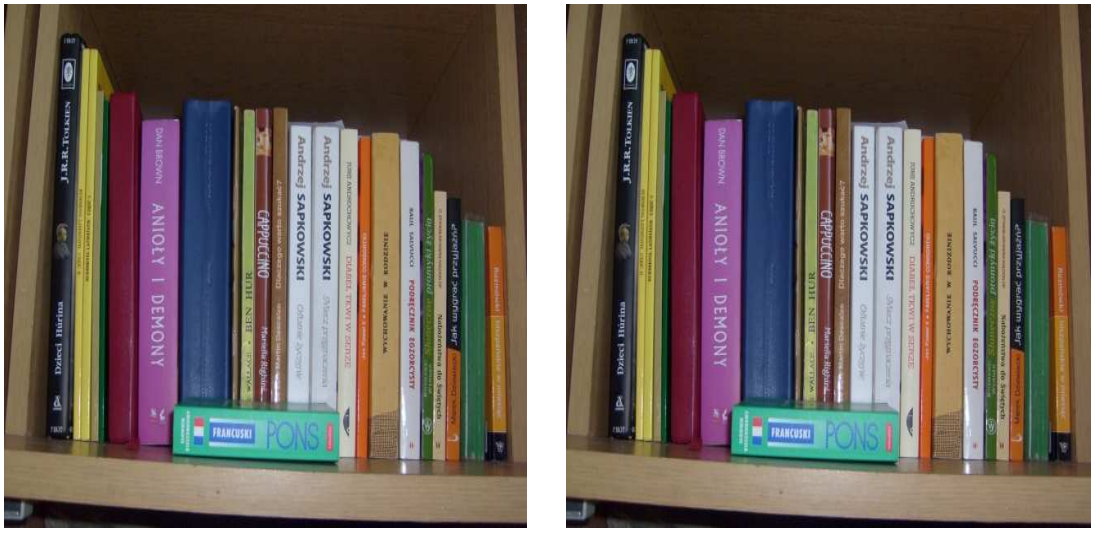
Sposobem nieznacznego polepszenia wyników estymacji dla źle wyliczonych pikseli jest użycie bloków o większych rozmiarach, jednak dla wielkości większych niż 12x12 pikseli zaczynają występować błędy złego skorelowania. Ponadto wiąże się to ze wzrostem czasu obliczeń. Dla całkowitego wyeliminowania niekorzystnych rezultatów można wektory różniące się znacząco od swojego sąsiedztwa, zastąpić medianą z otoczenia każdego z nich. Wtedy każdy z obliczonych obrazów będzie przedstawiał wektory skierowane w jednym kierunku. Taką sytuację przedstawia rysunek 4.20.



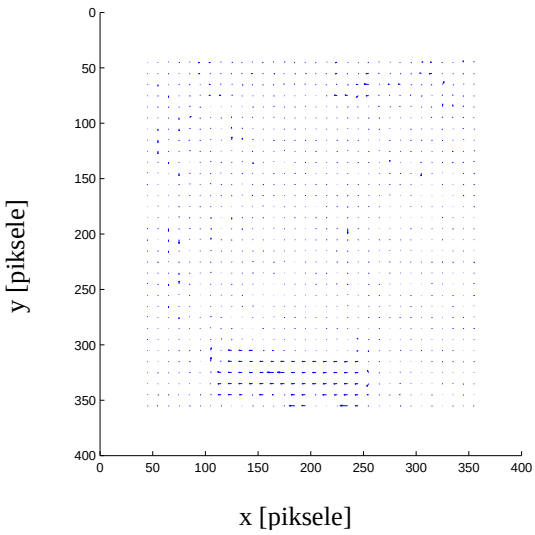
Rys. 4.21 Pole wektorów ruchu obliczone na podstawie rysunków 4.22. Zastosowanie mediany dla różniących się o więcej niż 0.5 piksela składowych dx i dy przesunięć.

Kolejny test wykonano na zdjęciach przedstawiających półkę z książkami. Różnicą pomiędzy fotografiami jest niewielkie przemieszczenie jednej z książek. Dobrano te obrazy tak, aby zawierały więcej szczegółów niż opisywane poprzednio. Przedstawiono je na rysunku 4.23. Przetestowano je algorytmem sympleksowym. Tym razem nie wystąpiły żadne artefakty i wyraźnie na rysunku 4.24 widoczne są wektory obrazujące przesunięcie książki. Stało się tak dlatego, że użyte obrazy nie zawierały jednolitych powierzchni, zastosowane było równomierne oświetlenie i nie były one poddane kompresji. Szkodliwy wpływ jej działania na estymację przepływu optycznego, można łatwo wykazać poprzez poddanie jednego z porównywanych obrazów kompresji. Efekt jaki wówczas występuje przedstawiono na rysunku 4.25.

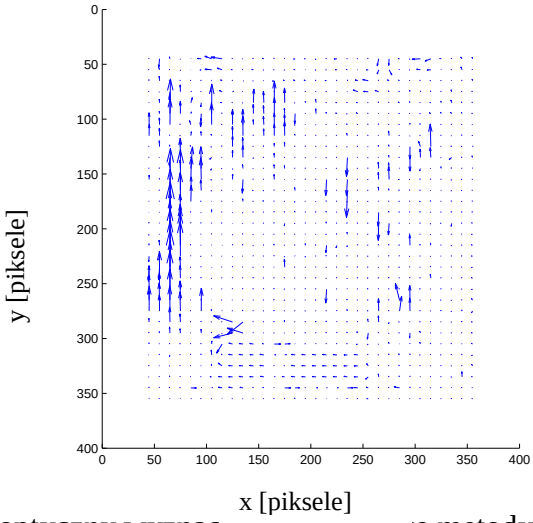
a) **b)**



Rys. 4.26 Zdjęcia półki z książkami: zielona książka podlega przesunięciu.



Rys. 4.27 Przepływ optyczny wyznaczony za pomocą metody sympleksowej



Rys. 4.28 Przepływ optyczny wyznaczony za pomocą metody sympleksowej, widoczny wpływ kompresji jednego z obrazów.

Podsumowując efekty działania badanych algorytmów stwierdzono, że mogą być stosowane przy wyznaczaniu przepływu optycznego na podstawie rzeczywistych obrazów. Zwrócono również uwagę, że domyślna kompresja podczas obróbki zdjęć, także działała niekorzystnie i uniemożliwiała uzyskanie poprawnych rezultatów, dlatego korzystano z plików nie skompresowanych.

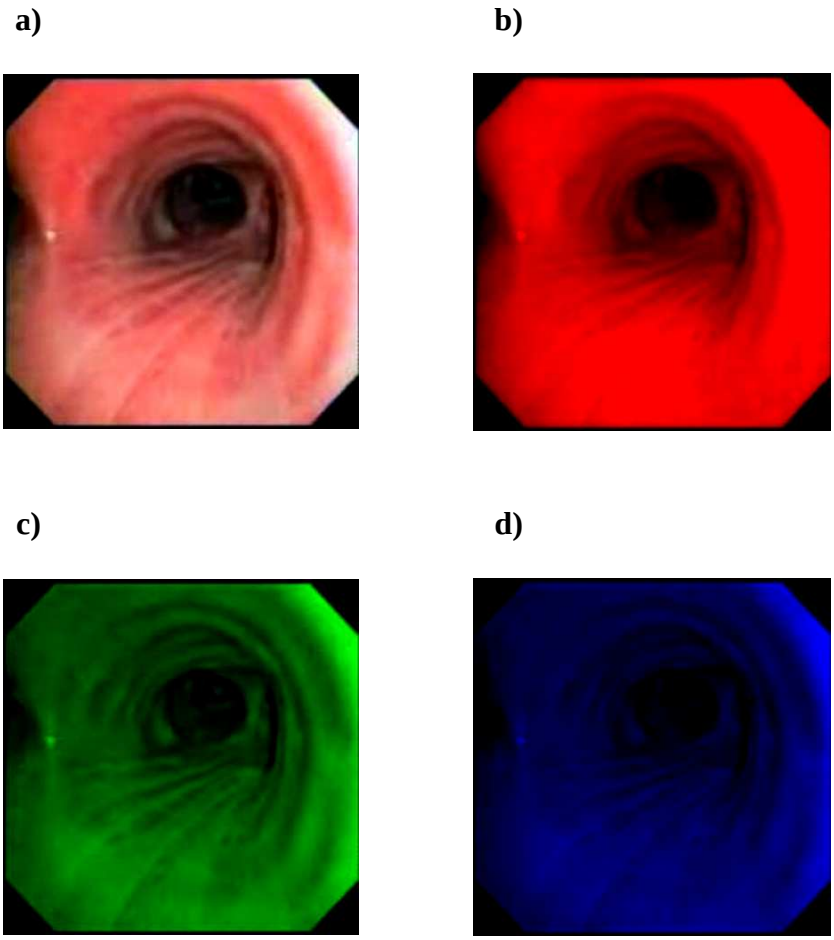
Czasy wykonywania algorytmów przedstawiały się niemal identycznie jak dla przypadku obrazów syntetycznych o dużych przesunięciach, dlatego pominięto ich prezentację w tym podrozdziale.

4.3 Wyznaczanie przepływu optycznego dla obrazów kolorowych

Wcześniejsze rozważania przedstawiały wyniki otrzymane na podstawie analiz obrazów monochromatycznych. W podrozdziale 4.3 zostaną zaprezentowane wyniki dla obrazów kolorowych. Każdy obrazek przedstawiony jest za pomocą odpowiedniej przestrzeni barwnej, która jest matematycznym opisem możliwych do wyświetlenia lub drukowania kolorów. Testowane obrazy, zaprezentowano w różnych przestrzeniach barwnych i ukazano ich podział na poszczególne składowe. Następnie pokazano, że obliczanie przepływu optycznego dla obrazów kolorowych sprowadza się do analizowania poszczególnych ich komponentów, dając przy tym bardzo dobre efekty. W testach zastosowano algorytm sympleksowy, który jak przedstawiono w poprzednich rozdziałach prezentował najlepsze efekty pod względem prędkości i jakości estymacji.

4.3.1 RGB

Najbardziej rozpowszechnionym modelem przestrzeni barw jest RGB. Na rysunku 4.30 przedstawiono obraz pochodzący z [12] i jego poszczególne składowe w omawianej przestrzeni. Zaproponowane rozwiązanie dla przestrzeni RGB polega na osobnej analizie poszczególnych składowych barwnych jak również obrazu przekonwertowanego z omawianej przestrzeni do odcieni szarości.



Rys. 4.29 Reprezentacja składowych obrazu w przestrzeni RGB: a) obraz oryginalny, b) składowa R, c) składowa G, d) składowa B.

Poszczególne składowe wczytywane do programu, są niczym innym jak macierzami jasności pikseli, dokładnie jak w obrazach monochromatycznych. Za to każda z nich prezentuje wartości odcieni jednego z kolorów RGB. Obliczanie przepływu optycznego, dla obrazów kolorowych, algorytmami prezentowanymi w tej pracy, może być zrealizowane poprzez konwersje danych wejściowych do jednego z monochromatycznych modeli barwnych. Poza użyciem poszczególnych modeli barwnych autor zastosował dwa inne sposoby:

- zsumowanie macierzy składowych i podzielenie ich przez trzy – obrazek monochromatyczny,
- transformacja do odcieni szarości poprzez użycie funkcji programu graficznego (zwykle polega ona na zsumowaniu z odpowiednimi wagami poszczególnych składowych).

Testy w tym podrozdziale dokonano na obrazach syntetycznych o większych przesunięciach badanych już w punkcie 4.1. Wybór ich był podyktowany, potrzebą sprawdzenia estymacji dla zróżnicowanych długości wektorów ruchu i koniecznością posiadania materiału referencyjnego, aby móc ocenić dokładność.

Rezultat pokazuje, że najlepsze efekty obliczania przepływu optycznego uzyskuje algorytm na składowej czerwonej i skali odcieni szarości. Można powiedzieć, że zastosowanie bardzo prostej metody i tak zapewniło dużą dokładność, biorąc pod uwagę duże przesunięcia występujące pomiędzy obrazami. Wyniki dla poszczególnych składowych, jak również porównanie ze sposobem obliczania średniej z poszczególnych komponentów i wynikiem otrzymanym na podstawie skali odcieni szarości przedstawiono w tabeli 4-13:

Tabela 4-13 Porównanie wyników obliczeń przepływu optycznego na różnych danych wejściowych

	Składowa R	Składowa G	Składowa B	Średnia Komponentów	Skala odcieni szarości
Błąd średni [piksele]	0.1575	0.1585	0.1587	0.1590	0.1577
Błąd maksymalny [piksele]	0.2890	0.2823	0.2821	0.3058	0.2743

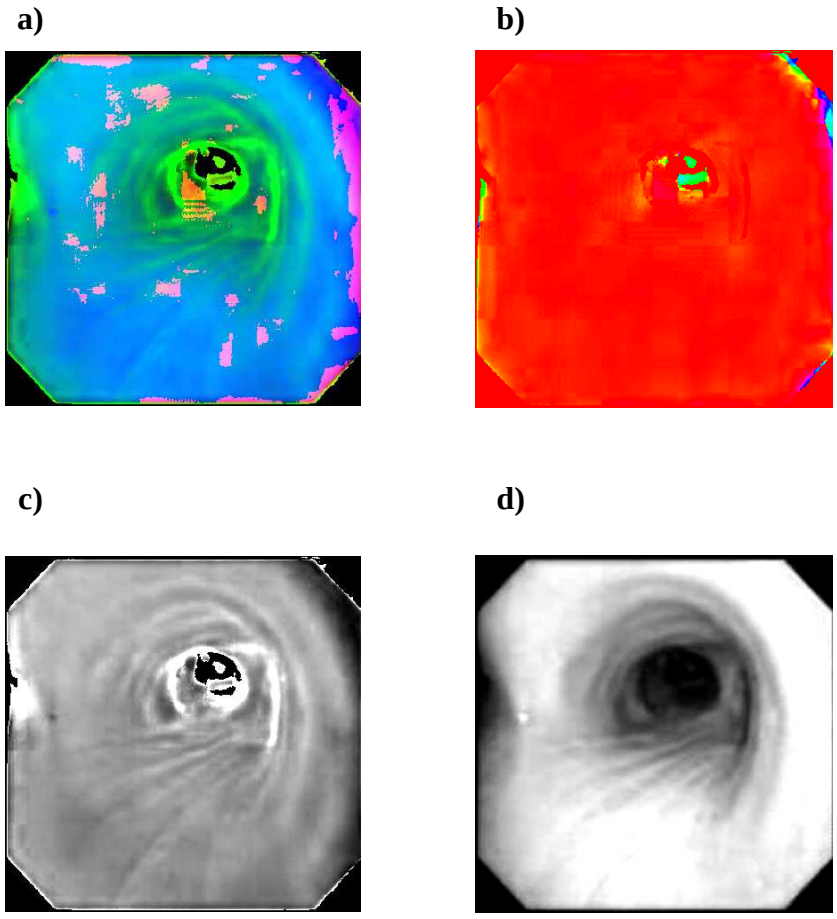
Na podstawie wyników można stwierdzić, że bardzo dobre efekty daje użycie poszczególnych składowych przestrzeni RGB. Obrazy pochodzące z bronchoskopu mają najwięcej odcieni koloru czerwonego, stąd najmniejszy błąd średni dla tej barwy. Najgorszą estymację spośród przedstawionych, zapewnia metoda obliczania średniej. Jednakże wyniki pokazują, że każdy ze sposobów zaprezentowanych w tabeli nadaje się do estymacji przepływu optycznego.

4.3.2 HSV

Prezentowany model opiera się na postrzeganiu barw jako światła padającego na przedmiot. Wszystkie odcienie wywodzą się od światła białego, którego widmo w kontakcie z obiektami w części ulega odbiciu, a w części jest absorbowane. Model jest przedstawiany jako stożek, którego podstawą jest koło barw. Nazwa tej przestrzeni pochodzi od określeń:

- H – częstotliwość światła (ang. *Hue*), opisana kątem na kole barw,
- S – nasycenie koloru (ang. *Saturation*), promień koła barw,
- V – moc światła białego (ang. *Value*), określana również literą B, będąca wysokością stożka.

Aby dokonać konwersji RGB na przedstawioną przestrzeń, skorzystano z funkcji Matlaba `rgb2hsv(obraz)`. Na rysunku 4.31 przedstawiono obraz i jego składowe w modelu HSV.



Rys. 4.30 Reprezentacja składowych obrazu w przestrzeni HSV: a) obraz w przestrzeni HSV, b) częstotliwość światła, c) nasycenie koloru, d) moc światła

Na podstawie obrazów przedstawionych na rysunku 4.30 można zaobserwować, że ze względu na charakterystykę modelu HSV, nie będzie możliwe używanie wszystkich jego składowych do estymacji pola wektorów ruchu, jak to wykonano w przestrzeni RGB. W zasadzie spośród przedstawionych składowych, obliczanie przepływu optycznego możliwe jest jedynie na ostatniej z nich – składowej V. Określa ona jasność, moc światła białego, co czyni ją najbardziej podobną do obrazów w odcieniach szarości. Wykonane na niej testy wykazały, że również na podstawie przestrzeni HSV istnieje możliwość estymacji przepływu optycznego. Wyniki zaprezentowano w tabeli 4-14.

Tabela 4-14 Przedstawienie wartości błędów dla składowej V, modelu HSV.

Blad średni [piksele]	0.1575
Błąd maks. [piksele]	0.2890

Uzyskane wartości błędów są identyczne jak dla składowej R modelu RGB. Niestety nie możliwe jest wykorzystanie innych komponentów przestrzeni HSV.

4.3.3 YUV

Z kolei YUV to model barw, w którym składowa Y charakteryzuje jasność obrazu, a pozostałe określają barwę. U jest przeskalowaną różnicą składowej B i Y, a V różnicą pomiędzy składową R i Y. Przechodzenie z przestrzeni RGB do YUV realizowane jest za pomocą wzorów:

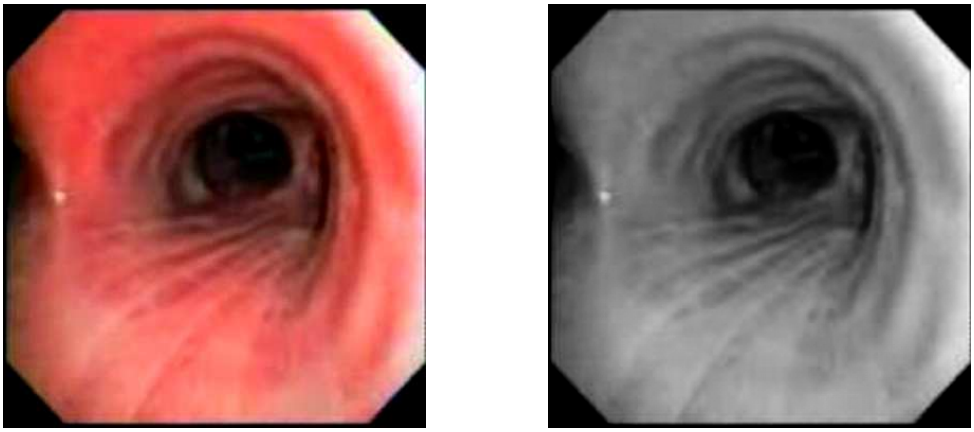
$$Y = 0.299R + 0.587G + 0.114B$$

$$U = 0.492\,(B - Y)$$

$$V = 0.877\,(R - Y)$$

Najwięcej treści z przedstawionych składowych niesie ze sobą luminancja. Na rysunku 4.31 widnieje obraz oryginalny i składowa Y.

a) **b)**



Rys. 4.31 Obraz oryginalny i jego składowa Y z modelu YUV

Procedura tworzenia obrazu Y ze składowych RGB jest często stosowana do generacji rysunku w skali odcieni szarości. Wynika z tego, że powinna ona również umożliwiać zastosowanie w algorytmach opisywanych w tej pracy. Otrzymane rezultaty potwierdzają tą tezę, co przedstawia tabela 4-15.

Tabela 4-15 Przedstawienie wartości błędów dla składowej Y, modelu YUV.

Blad średni [piksele]	0.1582
Błąd maks. [piksele]	0.2900

W analizowanym przypadku estymację ruchu na obrazach kolorowych można sprowadzić do estymacji na Y z modelu YUV, R z przestrzeni RGB lub V z HSV. Wykorzystanie jednej ze składowych modelu jaki reprezentuje obraz w zamian za wykonywanie operacji matematycznych służących przekształcaniu go do skali odcieni szarości jest czynnością bardziej oszczędną obliczeniowo. Ponadto niskie wartości błędów, mimo występujących dużych przesunięć pomiędzy obrazami, również świadczą o tym, że przepływ optyczny dla obrazów kolorowych może być obliczany za pomocą algorytmów działających na obrazach monochromatycznych.

Wnioski końcowe

W przedstawionej pracy zaimplementowane zostały algorytmy wyznaczania przepływu optycznego za pomocą metody siłowej i zoptymalizowanych. Zbadano możliwą do osiągnięcia dokładność, a przede wszystkim znacznie przyspieszono pierwotną wersję algorytmu, za pomocą licznych modyfikacji. Szybka i dokładna estymacja pola wektorów ruchu jest podstawą do stosowania takich technik, jak generacja obiektów 3D, na podstawie sekwencji obrazów. Algorytm ten został stworzony z myślą o wykorzystaniu go do tworzenia trójwymiarowych struktur obserwowanych przez kamerę video bronchofiberoskopu, które mają służyć pozycjonowaniu endoskopu w drzewie oskrzelowym i wspomaganiu lekarza podczas zabiegu. Dzięki zastosowany metodom uzyskano precyzyjne i szybkie algorytmy wyznaczania przepływu optycznego.

Pierwszym krokiem w stronę uzyskania pożądanej prędkości obliczeń, jest implementacja wszystkich algorytmów w języku C++. Podstawowym narzędziem wspierającym estymację jest hierarchiczna dekompozycja obrazów. Implementacja jej znacznie usprawnia działanie każdego opisanego algorytmu. Ponadto wpłynęła na dokładniejszą estymację pola wektorów metodą gradientową, jak również całkowicie zlikwidowała problem znajdowania minimów lokalnych algorytmem sympleksowym. Najbardziej czasochłonną procedurą jest interpolacja. W celu jej optymalizacji, przeprowadzono liczne testy, na podstawie których zostało wybrane podejście precyzyjne, ale również zapewniające najmniejszą złożoność obliczeniową. Modyfikacje w algorytmie generującym współczynniki jądra interpolacji, a także w samej funkcji interpolującej przyczyniły się do **pięciokrotnego przyspieszenia** całego algorytmu. Kolejnym krokiem stosowanym w celu przyspieszenia algorytmu było użycie bibliotek BLAS w dekompozycji obrazów.

Dodatkowo usprawniono algorytmy minimalizacji funkcji celu. W prezentowanej pracy opisano metodę gradientową i bezgradientową. Reprezentacją pierwszej grupy jest algorytm najszybszego spadku, który umożliwia redukcję czasu obliczeń, okupioną generacją większej ilości błędów. Ogólnie jednak metoda ta prezentuje również dużą szybkość i może być stosowana w wielu aplikacjach. Przedstawicielem drugiej grupy jest, występująca w zbiorze GSL, metoda sympleksów Neldera-Meada. Dzięki zależności jedynie od wartości funkcji celu, wykonuje ona jeszcze mniej iteracji niż strategia

gradientowa, przez co prezentuje największą efektywność pod względem czasowym spośród opisanych w tej pracy sposobów, co więcej generuje mniejsze błędy niż metoda siłowa.

Podsumowując, w pracy nad optymalizacją metody wyznaczania przepływu optycznego, uzyskano duży wzrost prędkości w porównaniu z początkową wersją wyszukiwania siłowego. Pierwotnie algorytm wykonywał się w ciągu 111 sekund. Zaproponowane optymalizacje i zastosowanie innych strategii poszukiwań, pozwoliły osiągnąć czas 0.16 sekund przy wykorzystaniu dekompozycji i metody Neldera-Meada. Dało to ponad sześćsetkrotne przyspieszenie. względem metody siłowej z r=0.1, na komputerze z jednordzeniowym procesorem Pentium M 1.86Ghz i 1.5GB pamięci RAM. Kolejne zwiększenie prędkości jest możliwe poprzez zrównoleglenie, pod kątem którego program został przygotowany. Ponadto znacznie poprawiono dokładność generowanych wyników. Na początku błąd średni przy obrazkach o niewielkich przesunięciach wynosił ok. 0.081 piksela. Zastosowanie precyzyjniejszej funkcji interpolującej, wielopoziomowości i algorytmu sympleksowego pozwoliło otrzymać błędy średnie o wartości 0.044 piksela. Zmniejszono przez to również błąd maksymalny o ponad sześćdziesiąt procent. Osiągnięta dokładność i szybkość stanowią dowód, że zaproponowane metody wyznaczania przepływu optycznego lub ich fragmenty znajdą zastosowanie w generacji trójwymiarowych struktur na bazie sekwencji obrazów 2D.

Ostatni podrozdział stanowi również potwierdzenie, że estymacja przepływu optycznego dla obrazów kolorowych jest możliwa poprzez transformację ich do jednego z monochromatycznych modeli barwnych. Można to wykonać na wiele sposobów. Jednym z nich jest przekształcenie obrazu do przestrzeni odcieni szarości. Innym podejściem jest zastosowanie jednej ze składowych modelu barwnego, do wyznaczania przepływu optycznego. Rezultaty przeprowadzonych testów wykazały, że każda z różnych przestrzeni kolorów może być w tym celu zastosowana: w modelu RGB do wyznaczania przepływu optycznego może służyć jedna ze składowych, w przestrzeni HSV wykorzystać można komponent V, a w YUV dokładną estymację zapewnia składowa Y.

Literatura

[1] Fleet David J., Weiss Y., *Handbook of Computer Vision: Optical Flow Estimation*, Springer, 2006

[2] http://of-eval.sourceforge.net/, maj 2008

[3] Smiatacz M., *Automatyczna lokalizacja i śledzenie obiektów na obrazie*, Politechnika Gdańska, 2005

[4] Trucco E., Verri A., *Introductory Techniques for 3-D Computer Vision*,

[5] Zieliński T., Bułat J., Socha M., Duplaga M., *Estymacja parametrów ruchu kamery bronchofiberoskopu metodą geometrii epipolarnej i przepływu optycznego z wykorzystaniem szybkiej zespolonej transformaty falkowej*, AGH, UJ, 2006

[6] Friedl L., Neil D., Pierce R. B., NASA Air Quality Program Team, *Air Quality Management Through Earth Observations & Models, EOM, Vol. XIV, No. 5*, lipiec 2005

[7] http://www.mathworks.com/products/viprocessing/demos.html, maj 2008

[8] http://www.mpeg.org/, maj 2008

[9] http://www.fxguide.com/article333.html, maj 2008

[10] Keys R. G., *Cubic Convolution Interpolation for Digital Image Processing*, IEEE, grudzień 1981

[11] Zieliński T., *Cyfrowe przetwarzanie sygnałów. Od teorii do zastosowań*, WKŁ, 2005

[12] http://www.youtube.com/watch?v=HH-NmeWeJts, maj 2008

[13] http://mathworld.wolfram.com/ExhaustiveSearch.html, maj 2008

[14] Borkowski D., materiały dydaktyczne: *Łatwy przykład optymalizacji metodą największego spadku*, AGH, 2002

[15] Nelder J. A., Mead R., *A simplex method for function minimization*, Computer Journal Vol. 7, 308-315, 1965

[16] http://optymalizacja.w8.pl/simplexNM.html, maj 2008

[17] Vernon D., *Fourier Vision – Segmentaion and Velocity Measurement using the Fourier Transform*, Kluwer Academic Publisher, 2001

[18] Bernard C., praca doktorska: *Wavelets and ill-posed problems: optic flow estimation and scattered data interpolation*, Ecole Polytechnique, Francja, 1999

[19] W. Press., Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, 1992

Dodatek A. Opis funkcji interpolujących

Wszystkie napisane przez autora kody źródłowe, dołączone są do pracy. Poniżej znajduje się opis ważniejszych funkcji używanych w programie. W tym punkcie opisane będą procedury związane z interpolacją.

```
void interBICUBIC_Flat_8x8_Fast( imgOutput *imgOUT, const imgInput
*imgIN, const sPoints *imgSP, const intKernel *kernel )
```

Zoptymalizowana wersja interpolacji bikubicznej działająca na blokach. Wyszukiwanie jądra interpolacji jest przeniesione poza obszar procedury interpolującej.

```
void interBICUBIC_Flat_8x8_Fast2( imgOutput *imgOUT, const imgInput
*imgIN, const sPoints *imgSP, const intKernel *kernel )
```

Najlepsza wersja interpolacji bikubicznej działającej na blokach. Poza główną pętlę jest przeniesione wyszukiwanie wierzchołka wycinka bloku mnożonego z maską interpolującą. Ponadto w strukturze imgSP przesyłane są jedynie dwa parametry – współrzędne wierzchołka wycinka.

```
void interBICUBIC_Flat_8x8_CBLAS( imgOutput *imgOUT, const imgInput
*imgIN, const sPoints *imgSP, const intKernel *kernel )
```

Wersja interpolacji bikubicznej działającej na blokach, wykorzystując funkcje CBLAS: `cblas_sdot`, która mnoży dwa wektory, a następnie dodaje ich elementy.

```
void interBICUBIC_Flat_8x8_CBLAS2( imgOutput *imgOUT, const imgInput
*imgIN, const sPoints *imgSP, const intKernel *kernel )
```

Wersja interpolacji bikubicznej działającej na blokach, wykorzystując funkcje CBLAS: `cblas_sdot`. Konstrukcja funkcji jest zmieniona, aby były możliwe operacje na większych blokach, przez co zastosowanie CBLAS jest bardziej efektywne.

```
void interBICUBIC_Flat_8x8_CBLAS3( imgOutput *imgOUT, const imgInput
*imgIN, const sPoints *imgSP, const intKernel *kernel )
```

Wersja interpolacji bikubicznej działającej na blokach, wykorzystując funkcje CBLAS drugiego lub trzeciego poziomu. Konstrukcja funkcji umożliwia jedno wywołanie funkcji CBLAS w celu optymalizacji. W zależności od wyboru stosuje się: `cblas_sgemv` – służące do mnożenia macierzy przez wektor, a także `cblas_sgemm` – mnożenie macierzy.

```
void findKernelBICUBIC( float *kerCoeff, float koffsetx, float koffsety )
```

Obliczanie jądra interpolacji według bezpośredniej implementacji wzoru (14).

```
float Rx( float x )
```

Implementacja wzorów matematycznych dla różnych wersji interpolacji bukubicznej. Funkcja wykorzystywana w `findKernelBICUBIC`.

```
void findKernelBICUBIC_2(float *kerCoeff, float koffsetx, float koffsety)
```

Obliczanie jądra interpolacji w szybszy sposób, metoda charakterystyczna dla funkcji interp2 Matlaba.

```
float polyx( float x )
```

Obliczanie wielomianów stosowanych w funkcji `findKernelBICUBIC_2`.

Dodatek B. Opis funkcji stosowanych w obliczeniach przepływu optycznego

Poniżej znajduje się krótki opis najważniejszych funkcji używanych w programach do wyznaczania pola wektorów ruchu.

```
void MVSearch( const char *filenameImg1, const char *filenameImg2, const kernelType kType )
```

Funkcja obliczająca przepływ optyczny, inicjalizuje większość zmiennych, zawiera procedury wczytywania obrazów, siatek poszukiwanych wektorów, główną pętlę programu w której możliwy jest wybór jednej z metod. Po wykonaniu wszystkich iteracji możliwy jest zapis współrzędnych obliczonych wektorów do pliku.

```
void MVSearch_decomp( const char *filenameImg1, const char *filenameImg2, const kernelType kType )
```

Funkcja obliczająca przepływ optyczny, podobna do `MVSearch`. Dodatkowo występuje w niej procedura dekompozycji obrazów przed rozpoczęciem wyszukiwania wektorów.

```
void MVSearch_threads( const char *filenameImg1, const char *filenameImg2, const kernelType kType )
```

Funkcja obliczająca przepływ optyczny, podobna do `MVSearch`. Dodatkowo występuje w niej podział fragmentów obrazka pomiędzy nowo utworzone wątki.

```
void initKernel( intKernel *kernel, const kernelType kType )
```

Inicjalizacja jądra interpolacji dla algorytmu siłowego. Wobec stałej siatki możliwych przesunięć w tej metodzie możliwa jest inicjalizacja wszystkich możliwych masek na początku algorytmu, poza główną pętlą.

```
void initKernel_var( intKernel *kernel, const kernelType kType, float koffsetx, float koffsety )
```

Inicjalizacja jądra interpolacji dla dowolnych przesunięć, określonych przez parametry `koffsetx` i `koffsety`.

```
void exhaustiveMVSearch_Flat( mVector *mv, mVector *mvOrg, const imgInput *img1, const imgInput *img2, intKernel *kernel, float half_range )
```

Funkcja wyszukiwania jednego wektora ruchu metodą siłową. Zawiera generację siatki wszelkich możliwych przesunięć, a następnie główną pętlę zajmującą się znajdywaniem współrzędnych prędkości przemieszającego się punktu, za pomocą metody korelacyjnej.

```
void exhaustiveMVSearch_Flat2( mVector *mv, mVector *mvOrg, const imgInput *img1, const imgInput *img2, intKernel *kernel, float half_range )
```

Funkcja podobna do `exhaustiveMVSearch_Flat`. Różni się brakiem procedury generującej strukturę poszukiwanych przesunięć dla grupy pikseli.

```
void calculatePatchSP( const mVector *mv, sPoints *patch, const float dx, const float dy )
```

Funkcja zajmująca się generacją struktury wypełnionej nowymi, poszukiwanymi pozycjami, które następnie przekazywane są do procedury interpolującej. Ostatecznie fukncja usunięta, ponieważ do przemieszczenia całego bloku o to samo przesunięcie wystraczą dwa parametry.

```
float calculateFit( const float *patch2, const float *patch1, const sGrid *sg )
```

Obliczenie sumy kwadratów różnic pomiędzy pikselami bloków wyciętych z obrazów.

```
float calculateFit_CBLAS( const float *patch2, const float *patch1, const
sGrid *sg )
```

Obliczenie sumy kwadratów różnic pomiędzy pikselami bloków za pomocą funkcji CBLAS: `cblas_scopy` (kopiowanie wektorów), `cblas_saxpy` (dodawanie wektorów i mnożenie przez liczbę) i `cblas_sdot` (mnożenie wektorów i sumowanie ich elementów). Funkcja nie wykorzystywana ze względu na większą złożoność niż procedura bez wspomagania BLAS.

```
void findAndSetBestFit( mVector *mv, const sGrid *sg, const int gridSize
)
```

Szukanie przesunięcia o jak najmniejszej wartości wskaźnika jakości zwracanego przez `calculateFit. Przemieszczenie to traktowane jest jako nowy wektor ruchu.`

```
void desiredPatch( float *patch2, const imgInput *img2, const mVector
*mv, const sGrid *sg )
```

Funckja wycina blok piskeli, którego wierzchołek jest równy kolejnemu punktowi na wczytanej siatce.

```
mVector * initGrid( int *gridSize, const intKernel *kernel )
```

Inizcjalizacji siatki pola wektorów ruchu. Jest ona tworzona w Matlabie i zapisywana do pliu, który jest czytany przez program w C.

```
void initIMGs( imgInput *img1, const char *filenameImg1, imgInput *img2,
const char *filenameImg2, const intKernel *kernel )
```

Inizcjalizacji obrazów. Wczytanie obu rysunków za pomocą VTK. Jeżeli danymi wejściowymi są obrazy kolorowe to konwertowane są do jednego z modeli monochromatycznych.

```
void decomposition(imgInput **imgD,int level)
```

Wykonanie dekompozycji na poziomie wskazanym przez `level`. Wyniki poszczególnych etapów zapisaywane są w tablicy `imgD`.

```
void chooseWavelet(lCoeffs *wCoeff,int number)
```

Wybranie falki wykorzystywanej w dekompozycji. 1 – db1, 2 – db2.

```
void lwt2(imgOutput *img,lCoeffs *wCoeff,int level)
```

Wykonanie dekompozycji na jednym poziomie. Zastosowanie szybkiej predykcyjnej transformaty falkowej.

```
float * lsupdate(int option,float *x,float *F,int DF,int xS,int yS)
```

Filtracja predykcja-uaktualnienie występująca w `lwt2`. W zależności od danych wejściowych, wykonuje operacje w jednym wymiarze: na kolumnach lub wierszach

```
void lwt2_CBLAS(imgOutput *img,lCoeffs *wCoeff,int level)
```

Funkcja posiadająca to samo zadaniae co `lwt2`, dodatkowo zoptymalizowana jest ona przez użycie CBLAS: `cblas_sdot`.

```
float * lsupdate_CBLAS(int option,float *x,float *F,int DF,int xS,int yS)
```

Funkcja posiadająca to samo zadaniae co `lsupdate`, dodatkowo zoptymalizowana jest ona przez zastosowanie wsparcia CBLAS: : `cblas_sdot`.

```
void *Pthread_fun( void *ptr )
```

Funkcja wywoływana przez nowo stworzony wątek. Przekazywane są do niej parametry wskazujące jaki fragment obrazka ma być przez nią przetwarzany.

Dodatek C. Skrytpy języka Matlab

Program Matlab jest cennym narzędziem służącym weryfikacji wyników, jak również generowaniu materiału testowego dla aplikacji napisanych w C++. Poniżej znajduje się kilka przykładowych m-plików.

Tabela C.1 Funkcja wyszukiwania siłowego.

```
function mv = exhaustiveMVSearch(mv, mvOrg, img1,img2, half_range)

	sg.X = 8; sg.Y = 8;
	WORSTFIT = 1e6; i = 1;
	for x = -half_range:0.1:half_range
		for y = -half_range:0.1:half_range
			sg.dx(i) = x;
			sg.dy(i) = y;
			i=i+1;
		end
	end
	gridSize = i-1;
	patchSP.X = sg.X; patchSP.Y = sg.Y;
	patch2 = desiredPatch(img2, mvOrg, sg );

	for i = 1:gridSize
		if( isPatchPositionPossible( img1, mv, sg.dx(i), sg.dy(i), sg.X,
		sg.Y ) == 0 )
			sg.fit(i) = WORSTFIT;
			continue;
		end
		val1 = mv.x + sg.dx(i);
		val2 = mv.y + sg.dy(i);
		ind1 = floor(val1); ind2 = floor(val2);

		[imgspx,imgspy] = calculatePatchSP(patchSP, val1-ind1, val2-ind2 );
		[ imgINspx, imgINspy ] = meshgrid( 0:9, 0:9   );
		im = img1.img( ind1:ind1+sg.X+1, ind2:ind2+sg.Y+1 );
		patch1 = interp2(imgINspx, imgINspy, im, imgspx, imgspy, 'cubic' );
		sg.fit(i) = calculateFit( patch2, patch1, sg );
	end
	mv = findAndSetBestFit(mv, sg, gridSize );
```

Tabela C.2 Funkcja dokonująca dekompozycji.

```
function OUT = decomposition(img,level);

	image.img = img.img( 1+img.OX:img.X-img.OX, 1+img.OY:img.Y-img.OY );
	rzadP=2;
	rzadU=2;

	for i=1:level+1
		OUT(i).img = zeros( img.X-2*img.OX, img.Y-2*img.OY);
	end

	OUT(4).img = image.img;
	OUT(4).X = img.X;
	OUT(4).Y = img.Y;

	for i=1:level
		[w,k]=size(image.img);
		wave = liftwave('db1');
		[AA( 1:w/2, 1:k/2 ), AA(1:w/2, k/2+1:k ), AA(w/2+1:w, 1:k/2
		), AA(w/2+1:w, k/2+1:k )] = lwt2(image.img,wave);

		image.img = AA( 1:w/2, 1:k/2 );

		image.X = (img.X-2*img.OX)/(2^i)+2*img.OX;
		image.Y = (img.Y-2*img.OY)/(2^i)+2*img.OY;
		imgout = zeros(image.X,image.Y);
		imgout( 1+img.OX:image.X-img.OX, 1+img.OY:image.Y-img.OY ) =
		image.img;

		OUT(4-i).img(1:w/2+2*img.OY,1:k/2+2*img.OX) = imgout;
		OUT(4-i).X = image.X;
		OUT(4-i).Y = image.Y;
	end
```

Tabela C.3 Podział na składowe przestrzeni RGB i HSV.

```
clear all;
close all;

filenameIN = 'data/image.png';
[im, map] = imread( filenameIN );
disp('Wymiary obrazu:');  [W, K, N] = size(im)

figure;
subplot(2,2,1); imshow(im); title('OBRAZ');

clear D; D(W,K,3)=uint8(0); D(:,:,1)=im(:,:,1);
subplot(2,2,2); imshow(D); title('Składowa RED');

clear D; D(W,K,3)=uint8(0); D(:,:,2)= im(:,:,2);
subplot(2,2,3); imshow(D);  title('Składowa GREEN');

clear D; D(W,K,3)=uint8(0); D(:,:,3)=im(:,:,3);
subplot(2,2,4);  imshow(D);  title('Składowa BLUE');

[hs] = rgb2hsv(im);

figure;
subplot(2,2,1); imshow(hs); title('OBRAZ');

subplot(2,2,2), colormap(hsv), imshow(hs(:,:,1));  title('Składowa H');

sat(:,:,3) = hs(:,:,2); sat(:,:,2) = hs(:,:,2); sat(:,:,1) = hs(:,:,2);
subplot(2,2,3); imshow(sat); title('Składowa S');

val(:,:,3) = hs(:,:,3); val(:,:,2) = hs(:,:,3); val(:,:,1) = hs(:,:,3);
subplot(2,2,4);  imshow(val); title('Składowa V');
```

Tabela C.4 Funkcja celu wykorzystywana w algorytmie najszybszego spadku.

```
function [J,patch1] = cryt2(x,imf,patch2,mv)

    dx = x(1);
    dy = x(2);
    val1 = mv.x+dx;
    val2 = mv.y+dy;
    ind1 = floor( val1 );
    ind2 = floor( val2 );

    im = imf(ind1:ind1+10, ind2:ind2+10 );

    x_ind = val1-ind1;
    y_ind = val2-ind2;

    [ imgINspx, imgINspy ] = meshgrid( 0:10, 0:10   );
    [imgspx,imgspy] = meshgrid( y_ind:7+y_ind, x_ind:7+x_ind );

    patch1 = interp2(imgINspx, imgINspy, im, imgspx, imgspy, 'cubic' );
    J = sum(sum((patch2-patch1).^2));
```

Tabela C.5 Funckja obliczająca gradient w metodzie najszybszego spadku

```
function xi = grad4(x, patch2, img, J)

    ind = 9;
    ind_2 = 7;
    [ imgINspx, imgINspy ] = meshgrid( 0:ind, 0:ind  );

    xp(1)=x(1);
    xp(2)=x(2);


    p(1,1)=x(1)+0.1;
    p(1,2)=x(2)-0.1;

    p(2,1)=x(1)+0.1;
    p(2,2)=x(2);

    p(3,1)=x(1)+0.1;
    p(3,2)=x(2)+0.1;

    p(4,1)=x(1);
    p(4,2)=x(2)+0.1;

    p(5,1)=x(1)-0.1;
    p(5,2)=x(2)+0.1;

    p(6,1)=x(1)-0.1;
    p(6,2)=x(2);

    p(7,1)=x(1)-0.1;
    p(7,2)=x(2)-0.1;

    p(8,1)=x(1);
    p(8,2)=x(2)-0.1;

    imgspx = ones(ind-1,ind-1,8);
    imgspy = ones(ind-1,ind-1,8);

    for i=1:8
        patch0(:,:,i) = img( floor(p(i,1)):floor(p(i,1)) + ind,
             floor(p(i,2)):floor(p(i,2)) + ind );
        l2=p(i,2)-floor(p(i,2));
        l1=p(i,1)-floor(p(i,1));

        [imgspx(:,:,i),imgspy(:,:,i)]=meshgrid( l2:ind_2+l2, l1:ind_2+l1 );
        patch1(:,:,i) = interp2(imgINspx, imgINspy, patch0(:,:,i),
             imgspx(:,:,i),imgspy(:,:,i),'cubic');

        dJ(i) = J-sum(sum((patch2-patch1(:,:,i)).^2));
    end

    [val,ind] = max(abs(dJ));
    Xi = [p(ind,1)-xp(1),p(ind,2)-xp(2)];
```

Dodatek D. Zawartość dołączanej płyty CD-ROM

Na dołączonym do pracy nośniku istnieje następująca struktura katalogów:

/src

W folderze umieszczono źródła programów języku C++

/data

Zawiera obrazy testowane w pracy

/grid

Zawiera siatki wektorów używane przy różnych konfiguracjach algorytmu, jak również współrzędne wektorów wzorcowych

/matlab

W katalogu umieszczono wszystkie m-pliki używane w pracy

/vec_out

Współrzędne wektorów wygenerowanych przez programy w C++

/projekt

Katalog zawiera cały projekt programu kDevelop z plikami źródłowymi i materiałami testowanymi

Zawartość.txt

Plik opisujący zawartość nośnika

Dodatek E. CD-ROM