

**POLITECHNIKA WROCŁAWSKA**  
**WYDZIAŁ ELEKTRONIKI**

---

KIERUNEK: Automatyka i robotyka

SPECJALNOŚĆ: Komputerowe systemy zarządzania procesami produkcyjnymi

**PRACA DYPLOMOWA**  
**MAGISTERSKA**

**Wykrywanie ruchu za pomocą metod  
dopasowywania bloków**

**Motion detection by block matching**

**AUTOR:**  
Marcin Sałata

**PROWADZĄCY PRACĘ:**  
prof. dr hab. inż. Ewaryst Rafajłowicz

**OPIEKUN:**  
prof. dr hab. inż. Ewaryst Rafajłowicz

---

**WROCŁAW 2008**

## **SPIS TREŚCI**

---

<b>1.Wstęp.</b>	5
<b>2.Wykrywanie ruchu.</b>	7
<b>2.1 Zadanie wykrywania ruchu.</b>	7
<b>2.2 Przegląd metod wykrywania ruchu.</b>	10
<b>2.2.1 Metody oparte na przepływie optycznym.</b>	11
<b>2.2.2 Metody oparte na dopasowywaniu.</b>	13
<b>2.2.2.1 Metoda różnicowa.</b>	13
<b>2.2.2.2 Metoda stałego progu.</b>	17
<b>2.2.3 Metody filtrów kwadraturowych.</b>	18
<b>2.3 Problemy w wykrywaniu ruchu.</b>	18
<b>2.3.1 Problem apertury.</b>	18
<b>2.3.2 Problem zgodności.</b>	19
<b>2.3.3 Inne problemy.</b>	19
<b>2.4 Systemy wykrywania ruchu – przegląd literatury.</b>	19
<b>3.Wykrywanie ruchu za pomocą metod dopasowywania bloków.</b>	23
<b>3.1 Sieć wykrywania ruchu.</b>	23
<b>3.2 Podział klatki na bloki.</b>	25
<b>3.3 Współczynniki dopasowania bloków.</b>	27
<b>3.4 Wektor ruchu.</b>	30

---

<b>3.5 Algorytmy przeszukiwania.</b>	32
<b>3.5.1 Algorytm TSS.</b>	33
<b>3.5.2 Algorytm 4SS.</b>	36
<b>3.5.3 Algorytm LOGS.</b>	39
<b>3.5.4 Algorytm CS.</b>	41
<b>3.5.5 Algorytm HEXBS.</b>	43
<b>3.5.6 Algorytm NTSS.</b>	46
<b>3.5.7 Algorytm BBGDS.</b>	47
<b>3.5.8 Algorytm FBBMA.</b>	49
<b>3.5.9 Algorytm PSO-ZMP.</b>	56
<b>3.5.10 Algorytm PSA.</b>	60
<b>3.5.11 Algorytm BSPA.</b>	63
<b>3.5.12 Algorytm FEBMEA.</b>	65
<b>3.5.13 Algorytm VSS.</b>	67
<b>3.5.14 Algorytm CDS.</b>	69
<b>3.5.15 Algorytm AMT.</b>	71
<b>3.5.16 Algorytm DS.</b>	72
<b>3.6 Zastosowanie metod dopasowywania bloków do wykrywania ruchu przegląd literatury.</b>	74

---

<b>4.Implementacje wykrywanych algorytmów.</b>	76
--	----

<b>4.1 Implementacja algorytmu 4SS.</b>	76
<b>4.2 Implementacja algorytmu TSS.</b>	77
<b>4.3 Implementacja algorytmu CS.</b>	79
<b>4.4 Implementacja algorytmu LOGS.</b>	79
<b>4.5 Implementacja algorytmu BBGDS.</b>	80

---

**5.Wyniki testów.** 81

---

---

**6.Wnioski i podsumowanie.** 96

---

---

**7.Bibliografia.** 97

---

## 1. Wstęp

Napisaniu tej pracy przyświecały dwa główne cele. Po pierwsze pracę napisano w celu przybliżenia czytelnikowi algorytmów wykrywania ruchu za pomocą metod dopasowywania bloków. W pracy zebrano przegląd tego typu algorytmów opisanych w różnego rodzaju publikacjach (spis publikacji w rozdziale 7). Drugim celem było zaimplementowanie i przetestowanie wybranych algorytmów. Algorytmy tego typu mogą być stosowane w różnego rodzaju systemach wykrywania poruszających się obiektów, a także mogą być jednym z modułów systemu identyfikacji poruszających się obiektów. Dlatego też warto przetestować i poznać ich działania. Zwłaszcza, że mogą być stosowane do wykrywania ruchu w czasie rzeczywistym.

Przed autorem postawiono więc dwa zadania. Po pierwsze opisanie algorytmów wykrywania ruchu za pomocą metod dopasowywania bloków znajdujących się w różnego rodzaju publikacjach naukowych. Po drugie zaimplementowanie wybranych algorytmów i przeprowadzenie testów. Ogólnie można stwierdzić, że algorytmy będą testowane pod kątem liczby wykrywanych ruchomych bloków i ocenianie wizualnie.

W pracy opisano kilkanaście różnych algorytmów wykrywania ruchu za pomocą metod dopasowywani bloków. Znalazł się tu także opis kilku współczynników dopasowywania bloków. Zawarto także opis pięciu zaimplementowanych algorytmów.

Poniżej znajduje się krótki opis zawartości poszczególnych rozdziałów pracy:

**Rozdział 2.** W tym rozdziale zostało przedstawione zadanie wykrywania ruchu. Znajduje się tu także przegląd innych metod wykrywania ruchu niż wykrywanie za pomocą metod dopasowywania bloków. Ten rozdział zawiera także ciekawe (według autora) zastosowania różnego rodzaju systemów wykrywania ruchu. Rozdział ten zawiera także opis problemów spotykanych podczas wykrywania ruchu.

- Rozdział 3.** Ten rozdział zawiera opis algorytmów wykrywania ruchu za pomocą metod dopasowywania bloków. Rozdział ten jest poświęcony prezentacji m. in. różnego rodzaju współczynników dopasowywania bloków oraz algorytmów przeszukiwania. Ten rozdział zawiera definicje parametrów algorytmów wykrywania ruchu za pomocą metod dopasowywania bloków. Przedstawiono w nim także praktyczne zastosowania systemów wykorzystujące tego typu metody do wykrywania ruchu.
- Rozdział 4.** W tym rozdziale zaprezentowano zaimplementowane przez autora algorytmy wykrywania ruchu za pomocą metod dopasowywania bloków. Znajdują się tam wizualne rezultaty osiągane przez programy oraz uwagi na temat implementacji tego typu algorytmów.
- Rozdział 5.** Ten rozdział zawiera wyniki różnego rodzaju testów zaimplementowanych algorytmów.
- Rozdział 6.** W tym rozdziale znajdują się wnioski z przeprowadzonych testów i podsumowanie pracy.
- Rozdział 7.** W tym rozdziale przedstawiono literaturę, z której korzystał autor podczas pisania tej pracy.

## **2. Wykrywanie ruchu**

### **2.1. Zadanie wykrywania ruchu**

Ruch wykrywamy obserwując sekwencję obrazów (lub film, który też jest sekwencją obrazów). Obraz video jest dużo lepszym źródłem informacji od obrazu statycznego. Sekwencja obrazów pozwala określić dynamikę otoczenia. Termin ruch zwykle kojarzy się ze zmianami. Czym więc jest ruch na obrazie filmowym lub sekwencji obrazów? Jest zjawiskiem odkrywającym dynamikę zmian przestrzeni na obrazie. Więc aby wykryć ruch musimy porównać dwa obrazy. Jeden z nich będzie obrazem odniesienia a drugi obrazem badanym. Ogólnie można stwierdzić, że każda różnica pomiędzy obrazem odniesienia i obrazem badanym jest ruchem (należy jeszcze wziąć poprawkę na szumy i ruchy kamery).

Nasze oczy w połączeniu z mózgiem to dobry system wykrywania ruchu. Jesteśmy w stanie zaobserwować ruch na obrazie jaki do nas dociera. Lecz maszyna tak inteligentna nie jest. Musimy jej pokazać, które obiekty są w ruchu. Oczywiście do tego celu stosujemy odpowiednie algorytmy, które sprawią, że przy pomocy zwykłego domowego komputera będziemy w stanie wyśledzić poruszające się obiekty na filmie (bądź to z kamery bądź to z pliku video).

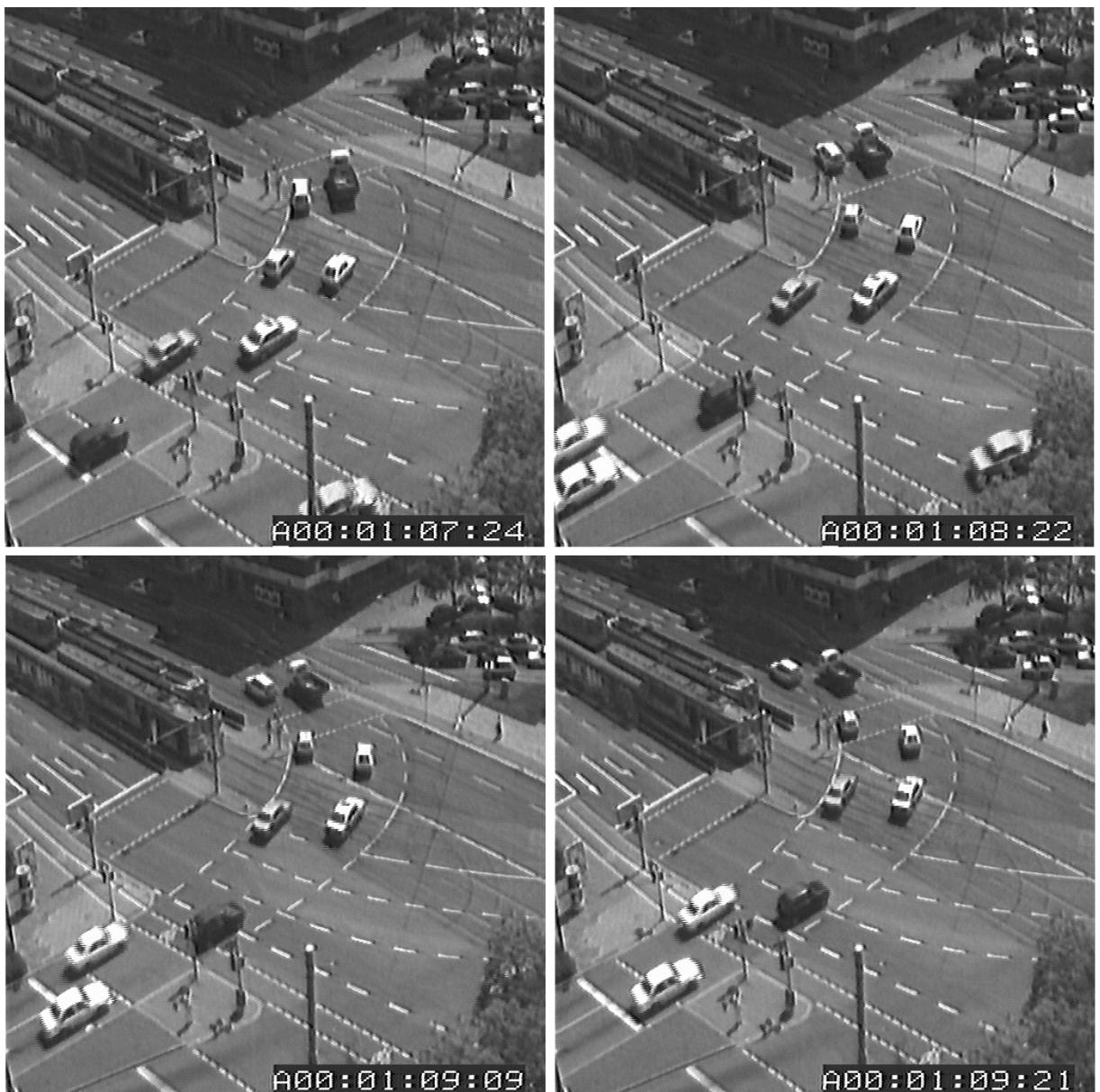
Jeszcze jakiś czas temu na wykrywanie ruchu w czasie rzeczywistym mogły sobie pozwolić wielkie firmy dysponujące odpowiednim sprzętem. Rozwój techniki komputerowej i algorytmów przetwarzania obrazów sprawiły, że jest to problem, którym może zająć się użytkownik „zwykły” komputera.

Wbrew pozorom techniki wyrywania ruchu mają niewiele wspólnego z klasycznym przetwarzaniem obrazów. Techniki przetwarzające sekwencje obrazów (z jakimi mamy do czynienia przy wykrywaniu ruchu) są inne niż te przetwarzające obrazy [17]. Ruch można wykrywać badając plik filmowy lecz można także to robić porównując ze sobą kolejne obrazy z sekwencji obrazów. To w jaki sposób będzie prowadzone wykrywanie ruchu zależy od zjawiska (środowiska) jakie badamy. Założmy, że wykrywamy ruch w jakimś pomieszczeniu. Może to być np. obraz z kamery przemysłowej. W tym przypadku musimy badać obraz video dlatego, że

wyniki wykrywania ruchu muszą być prezentowane na żywo. Inny przykład to analiza ruchu dotycząca zjawiska wolno zmieniającego się jakim może być badanie poruszających się planet (lub innych ciał niebieskich) z Ziemi. Badając takie zjawisko możemy obserwować niebo przechwytywać obraz z teleskopu z jakąś częstotliwością np. co 10 min. Uzyskamy w ten sposób sekwencję obrazów, która oczywiście może być pokazana jako film, ale zmieniające się obrazy tak naprawdę nie oddają rzeczywistego czasu. W jednym i drugim przypadku stosujemy te same techniki wykrywania ruchu.

Wykrywanie ruchu ma bardzo szeroki zakres zastosowań. Można stwierdzić, że systemy wykrywania poruszających się obiektów znajdują zastosowanie począwszy od użytku domowego a na przemyśle skończywszy. A pomiędzy tymi dwiema gałęziami jest wiele innych zastosowań. Oto niektóre z zastosowań wykrywania ruchu:

- biologia (np. śledzenie rozwoju procesów biologicznych)
- bezpieczeństwo (np. ochrona mienia)
- militaria (np. śledzenie celu)
- przemysł (np. robotyka)
- transport (np. wykrywanie pojazdów, statków itp.)
- meteorologia (np. wykrywanie poruszających się chmur, huraganów itp.)
- i inne



Rysunek 1. Na kolejnych obrazach sekwencji obrazów widać poruszające się obiekty (np. samochody)

Oczywiście systemy wykrywania ruchu odpowiadają tylko za wykrycie poruszających się obiektów. Nieistotne jest czy będą to liście poruszające się z wiatrem czy intruz w pomieszczeniu gdzie nie powinno być nikogo. Aby określić, które obiekty są ważne dla naszego systemu należy go wyposażyć w moduły odpowiedzialne za rozpoznawanie tychże obiektów. Oczywiście obiekty ważnymi są te, których ruch

mamy zamiar śledzić. Niniejsza praca jest wyłącznie o wykrywaniu ruchu i opisuje metody związane z wykrywaniem ruchu.

Aby wykryć ruch należy najpierw zarejestrować obraz, na którym należy wykryć ten ruch. Jeśli systemy wykrywania ruchu mają działać efektywnie to powinny pracować w czasie rzeczywistym. Nie zależy nam raczej na tym aby system wykrywania ruchu zwrócił nam obraz wyjściowy z zaznaczonymi ruchomymi obiekta po kilku minutach. Film odtwarzany jest z prędkością powyżej 20 fps (ang. frames per second) więc należałoby wymagać aby użytkownik na bieżąco obserwował poruszające się obiekty. A co za tym idzie system powinien w ciągu sekundy porównać ponad 20 par obrazów (obraz odniesienia i badany, czyli poprzedni oraz obecny). Algorytmy wykrywania ruchu radzą sobie z tym najprościej jak się da. Mianowicie wprowadzają pewne uproszczenia i uogólnienia.

## 2.2. Przegląd metod wykrywania ruchu

Techniki wykrywania ruchu dzielą się na trzy główne rodziny:

- oparte na optical flow (*ang. optical flow-based techniques*)
- oparte na dopasowywaniu (*ang. correspondence-based techniques*)
- techniki filtrów kwadraturowych (*ang. quadrature filter techniques*)

Metody wykrywania ruchu za pomocą metod dopasowywania bloków są tematem niniejszej pracy. Zostaną one dokładnie omówione w dalszej części. Chciałbym jednak przybliżyć pokrótko pozostałe metody.

Należy stwierdzić ogólnie, że wykrywanie ruchu to w jakimś stopniu wykrywanie zmian szarości pikseli na obrazie [17]. Przejeżdżający samochód powoduje zmianę wartości szarości poszczególnych pikseli w obszarach obrazu, w których się pojawia. Poniżej opiszę kilka najpopularniejszych metod wykrywania ruchu. Będzie to krótki przegląd tych metod.

### 2.2.1. Metody oparte na przepływie optycznym

Ogólnie można stwierdzić, że przepływ optyczny (*ang. optical flow*) polega na przepływie wartości szarości pikseli na obrazie [17].

Przyjmijmy, że wartości jasność punktu o współrzędnych  $(x, y)$  w chwili  $t$  jest oznaczana przez

$$I(x, y, t) \quad (1)$$

Postawmy dwa warunki:

1. Wartość jasności  $I(x, y, t)$  zależy od współrzędnych  $x$  i  $y$ .
2. Wartość jasności  $I(x, y, t)$  dla punktów nie poruszających się lub ruchomych nie zmienia się w czasie

Założymy, że po upłynięciu czasu  $dt$  punkt przemieścił się. Wartość przemieszczenia to  $(dx, dy)$ . Z szeregu Taylora możemy wyznaczyć wartość  $I(x+dx, y+dy, t+dt)$

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

Zgodnie z warunkiem 2 możemy stwierdzić, że

$$I(x + dx, y + dy, t + dt) = I(x, y, t) \quad (3)$$

oraz

$$\frac{\partial I}{\partial x} dx + \frac{\partial I}{\partial y} dy + \frac{\partial I}{\partial t} dt + \dots = 0 \quad (4)$$

Dzieląc równanie (3) przez  $dt$  i definiując

$$\frac{dx}{dt} = u \text{ oraz } \frac{dy}{dt} = v \quad (5)$$

otrzymujemy równanie

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

które znane jest jako **warunek przepływu optycznego**. Natomiast  $u$  oraz  $v$  to prędkości piksela odpowiednio w kierunku pionowym i poziomym.

O tym jak szybko zmienia się jasność danego piksela mówi  $\frac{\partial I}{\partial t}$ . Natomiast o przestrzennych zmianach jasności w obrazie mówią  $\frac{\partial I}{\partial x}$  oraz  $\frac{\partial I}{\partial y}$ .

Istnieje wiele technik wykrywania ruchu opartych na przepływie optycznym. Najważniejsze z nich to:

- **metoda różnicowa pierwszego rzędu** (ang. *first-order techniques*) – opisana w pracach [21] [22] [23] [24] [25].
- **metoda różnicowa drugiego rzędu** (ang. *second-order techniques*) – opisana w pracach [26] [27] [28].
- **metoda tensorowa** (ang. *tensor-based techniques*) – opisana szerzej w pracach [34] [35] [36] [37] [38].
- **metoda globalnych ograniczeń** (ang. *global constraints*) – więcej informacji o tej metodzie zawarto w pracach [29] [19] [26] [32] [33].

Funkcje wyliczające przepływ optyczny znajdują się w bibliotekach OpenCV. Bibliotekę można pobrać z <http://www.intel.com/technology/computing/opencv/>. W bibliotece OpenCV znajdują się następujące funkcje wyliczające przepływ optyczny:

- *CalcOpticalFlowHS* – funkcja wykorzystuje metodę Horn'a i Schunk'a

- *CalcOpticalFlowLK* – funkcja wykorzystuje metodę Lucas'a i Kanade'a
- *CalcOpticalFlowBM* – funkcja wykorzystuje metodę blokową
- *CalcOpticalFlowPyrLK* - funkcja wykorzystuje piramidową metodę Lucas'a i Kanade'a

## 2.2.2. Metody oparte na dopasowywaniu

### 2.2.2.1. Metoda różnicowa

W metodzie tej stosowany jest tzw. **obraz różnicowy**. Obraz ten jest obrazem binarnym. Wartości pikseli takiego obrazu przyjmują wartości 1 i 0. Wartość 1 oznacza, że istnieje dość znaczna różnica pomiędzy porównywanymi pikselami. Wartości  $d(i, j)$  takiego obrazu różnicowego wyznacza się zgodnie ze wzorem:

$$d(i, j) = \begin{cases} 0 \Leftrightarrow |f_1(i, j) - f_2(i, j)| \leq \varepsilon \\ 1 \Leftrightarrow |f_1(i, j) - f_2(i, j)| > \varepsilon \end{cases} \quad (7)$$

0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	1	0	1	0	0	1	0	1	0	0	0
0	0	1	1	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1	0
0	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	1	1	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1	0	1	0	0	1
1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	0	0	1	1	1	0	0	0	1	1	0	0
0	0	0	1	0	1	1	1	0	0	0	1	1	1	1	0	1	0	1	1	1	0	0	1	0	1	0
0	0	0	0	1	1	0	0	0	1	1	0	1	1	1	0	1	1	1	1	0	0	1	0	0	0	0
0	1	1	1	0	1	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0
0	0	1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	1	0	1	0	1	0	0	0	0
1	1	1	0	1	0	1	1	1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	1
0	0	0	1	1	0	1	0	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	0	1	0	0
1	1	1	1	0	0	1	0	1	0	0	0	1	1	1	1	1	1	0	1	0	1	0	0	1	0	0
1	1	0	0	0	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	1	0	1	1	0	0
0	0	1	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	1	1	1	0	1	0	0	1	1
0	1	0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	0	1	0	0
0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0
0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0

**Rysunek 2.** Przykładowy obraz binarny

Pola na obrazie różnicowym (binarnym) przyjmujące wartości 1 nie zawsze oznaczają ruch pikseli. Piksel na obrazie binarnym może przyjąć wartość 1 w następujących sytuacjach:

- $f_1(i, j)$  jest pikselem ruchomym a  $f_2(i, j)$  jest pikselem tła (lub odwrotnie)
- $f_1(i, j)$  jest pikselem ruchomym i  $f_2(i, j)$  jest pikselem ruchomym lecz oba piksele dotyczą dwóch różnych obiektów
- $f_1(i, j)$  jest pikselem ruchomym i  $f_2(i, j)$  jest pikselem ruchomym znajdującym się na tym samym obiekcie
- zakłócenia, szумy itp.

Należy jednak zwrócić uwagę na szумy jakimi obarczony jest obraz. Mogą one spowodować błędne obliczenie różnicy pomiędzy dwoma pikselami. Innym zjawiskiem na jakie należy zwrócić uwagę w tej metodzie jest kontrast pomiędzy tłem a obiekty.

Istnieje kilka modyfikacji tej metody. Stosuje się ją m. in. jako jedną z metod wykrywania ruchu za pomocą metod dopasowywania bloków [57][42].

Shinohara w [56] proponuje dwa algorytmy wykrywania ruchu oparte na metodzie różnicowej. Definiuje on obraz tła (*ang. background image*) i obraz wejściowy (*ang. input image*), przy pomocy których wykrywa poruszające się obiekty. Obrazem tła nazywa on obraz, na którym nie ma poruszających się obiektów. Użytkownik może sam określić jak taki obraz wygląda (tzn. może zdefiniować początkowy obraz tła). Obrazem wejściowym Shinohara nazywa obraz, na którym szukamy poruszających się obiektów. Shinohara proponuje metodę, w której w każdym kroku możliwe jest uaktualnienie obrazu tła. Aktualizacja obrazu tła przeprowadzana jest zgodnie ze wzorem:

$$I_{B(i,j)}(n) = A \cdot I_{IN(i,j)} + (1 - A) \cdot I_{B(i,j)}(n-1) \quad (8)$$

gdzie:

$I_{B(i,j)}(n)$  – uaktualniany piksel  $(i, j)$  obrazu tła

$A$  – współczynnik całkowania tła (*ang. background integration parameter*)

$I_{IN(i,j)}$  – piksel  $(i, j)$  obrazu wejściowego

$I_{B(i,j)}(n-1)$  – piksel  $(i, j)$  poprzedniego obraz tła

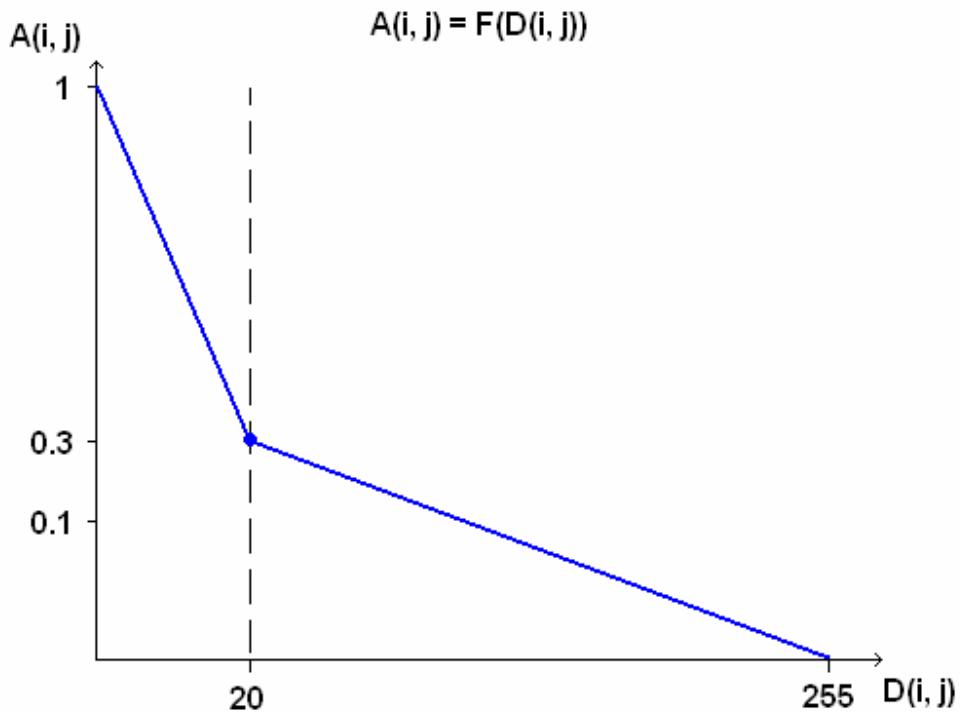
Parametr  $A$  jest stały dla każdego piksela obrazu tła. Przedstawiony przez niego algorytm (pierwszy) wygląda następująco:

**Krok 1:** Dla każdego piksela z obrazu wejściowego policz bezwzględną różnicę wartości szarości tego piksela i odpowiadającego mu piksela na obrazie tła.

- Krok 2:** Policz średnią wartość bezwzględnej różnicy (uzyskanej w kroku 2) dla niewielkiego obszaru pikseli w najbliższym otoczeniu danego piksela. Badany piksel musi znajdować się w centrum takiego obszaru.
- Krok 3:** Na podstawie wyliczonej średniej różnicy oceń czy piksel należy do tła czy do obiektu poruszającego się.
- Krok 4:** Uaktualnij obraz tła.

Natomiast drugi algorytm Shinohara opisuje następująco:

- Krok 1:** Dla każdego piksela z obrazu wejściowego policz bezwzględną różnicę wartości szarości tego piksela i odpowiadającego mu piksela na obrazie tła.
- Krok 2:** Zmień wartość współczynnika A zgodnie z wcześniej przyjętą zależnością (przykładową zależność pokazuje rysunek 3).
- Krok 3:** Policz średnią wartość bezwzględnej różnicy (uzyskanej w kroku 2) dla niewielkiego obszaru pikseli w najbliższym otoczeniu danego piksela. Badany piksel musi znajdować się w centrum takiego obszaru.
- Krok 4:** Na podstawie wyliczonej średniej różnicy oceń czy piksel należy do tła czy do obiektu poruszającego się.
- Krok 5:** Uaktualnij obraz tła.



**Rysunek 3.** Przykładowa zależność współczynnika  $A(i, j)$  dla piksela  $(i, j)$  od wartości bezwzględnej różnicy  $D(i, j)$  dla piksela  $(i, j)$

#### 2.2.2.2. Metoda stałego progu

Metoda stałego progu (*ang. fixed threshold*) jest jedną z najprostszych metod wykrywania ruchu [57]. Wykrywanie ruchu polega na obliczeniu wartości następującego współczynnika:

$$\rho_k(x) = I_k(x) - I_{k-1}(x) \quad (9)$$

gdzie:

$I_k(x)$  – wartość szarości piksela  $x = [x, y]^T$  z bieżącej klatki

$I_{k-1}(x)$  – wartość szarości piksela  $x = [x, y]^T$  z poprzedniej klatki

$\rho_k(x)$  – współczynnik odmiенноści dwóch pikseli

Następnie porównuje się współczynnik  $\rho_k(x)$  z przyjętym wcześniej progiem  $\theta$ . Jeśli jego wartość jest większa od przyjętego progu  $\theta$  tzn., że piksel  $x$  jest elementem ruchomego obiektu.

Aby zmniejszyć wpływ szumów na wartość współczynnika  $\rho_k(x)$  można zastosować następujące kryterium oceny dynamiki piksela:

$$\rho_k^2(x) \geq \theta \quad (10)$$

Jeśli powyższe kryterium jest prawdziwe tzn., że piksel jest elementem ruchomego obiektu.

Kryterium oceny dynamiki można stosować także do większych regionów np. bloków. Jeśli przyjmiemy, że taki obszar składa się z  $N$  punktów to kryterium oceny dynamiki będzie wyglądać następująco:

$$\frac{1}{N} \sum_{i=1}^N \rho_{k_i}^2(x) \geq \theta \quad (11)$$

### 2.2.3. Metody filtrów kwadraturowych

Metody wykorzystujące filtry kwadraturowe zostały szerzej opisane w pracach [38] [39] [40] [41].

Widmo Fouriera poruszających się obiektów tworzy pewną płaszczyznę, a metody filtrów kwadraturowych (*ang. quadrature filter technique*) przy użyciu filtrów kwadraturowych oceniają na podstawie tej płaszczyzny prędkość obiektów.

## 2.3. Problemy w wykrywaniu ruchu

### 2.3.1. Problem apertury

Problem apertury (*ang. aperture problem*) ma miejsce w sytuacji gdy porównujemy dwa obrazy [17][2]. Jeśli punkt (bądź obiekt) z obrazu badanego znajduje się blisko

krawędzi lub któregoś z rogów obrazu to tworzy się tzw. szczelina (*ang. aperture*). To powoduje, że nie można porównać punktu z obrazu odniesienia z punktem z obrazu badanego. Dlatego, że na obrazie badanym punkt, który chcielibyśmy wziąć pod uwagę znajduje się poza klatką. Innymi słowy – szerokość szczeliny jest mniejsza od przesunięcia z jakim algorytm porównuje ze sobą odpowiednie piksele. Oczywiście problem ten zachodzi tylko dla obiektów ruchomych.

### 2.3.2. Problem zgodności

Problem apertury jest szczególnym przypadkiem problemu zgodności (*ang. correspondence problem*) [17]. Ogólnie można powiedzieć, że ma on miejsce w sytuacji gdy nie możemy znaleźć punktu z obrazu odniesienia na obrazie badanym. Jeśli odniesiemy się do metod wykrywania ruchu za pomocą dopasowywania bloków, to problem ten ma miejsce gdy bloki są zbyt małe w stosunku do rozmiaru obiektów. W takiej sytuacji mimo, że obiekt się porusza system nie wykryje bloków znajdujących się wewnątrz poruszającego się obiektu.

### 2.3.3. Inne problemy

Innymi zjawiskami, które zakłócają wykrywanie ruchu są wszelkiego rodzaju szумy.

Najczęstsze szumy to:

- niezamierzone ruchy kamery
- małe obiekty, które przysłaniają obiekt (np. śnieg, deszcz itp.)
- tło, na którym znajdują się badane obiekty
- i inne

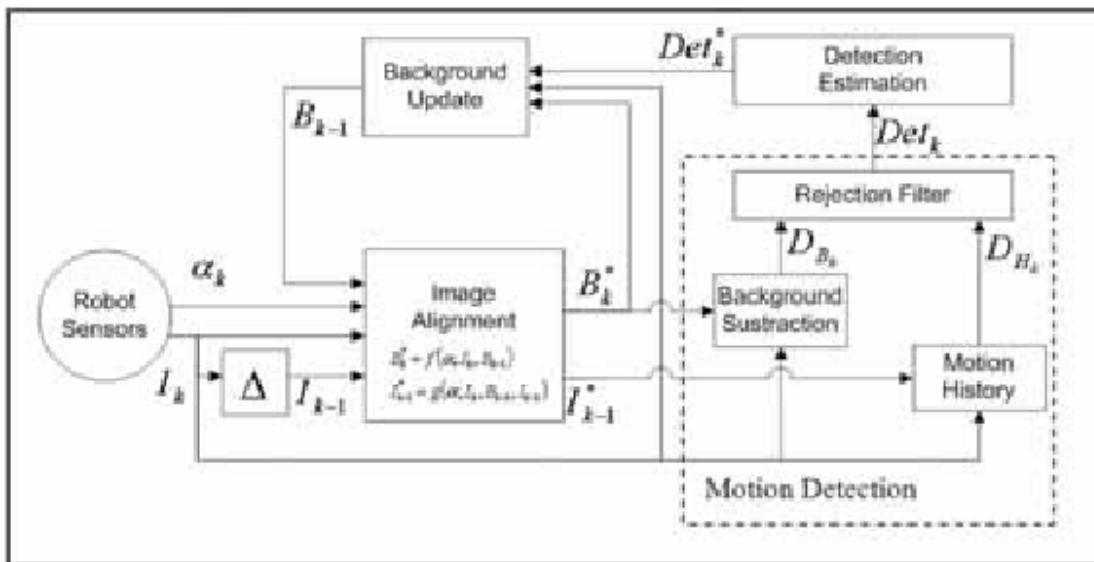
## 2.4. Zastosowanie wykrywania ruchu – przegląd literatury

W pracy [59] Jencik, Planta i Conradt przedstawiają projekt detekcji ruchu w systemie nawigacyjnym opartym na wzroku muchy. System ten ma za zadanie wyznaczenie kierunku poruszania się latającego robota. Robot posiada dwie kamery

o kącie widzenia wynoszącym  $56^\circ$ . Robot posiada moduł komunikacji bezprzewodowej, który komunikuje się z komputerem. Aplikacja na podstawie obrazów otrzymywanych z kamer zajmuje się detekcją ruchu obiektów wokół robota. Taki system będzie sercem układu nawigacyjnego tego robota.

Moore w [77] opisuje system wykrywania i rozpoznawania poruszających się obiektów. Obiektami tymi są ludzie. Zastosowanie takiego systemu jest bardzo szerokie. Oczywiście jego głównym zadaniem jest poprawa bezpieczeństwa. Autor sugeruje, że może on być stosowany na lotniskach, w muzeach a nawet w kasynach. Innym zastosowaniem dla takiego systemu może być motoryzacja. W motoryzacji taki system mógłby poprawić bezpieczeństwo pieszych poprzez szybkie reagowanie na nagle pojawiających się ludzi na drodze jadącego samochodu.

W pracy [78] Solar i Vallejos przedstawiają system wykrywania poruszających się obiektów stosowany w robocie piłkarskim AIBO. Schemat blokowy tego systemu przedstawiono na rysunku 4 (ilustracja pochodzi z pracy [78]).



**Rysunek 4.** Schemat blokowy systemu wykrywania poruszających się obiektów w robocie piłkarskim.

Sensorem ruchu jest oczywiście kamera. Obraz trafia do kolejnych bloków systemu. System składa się z czterech podsystemów: *Image Alignment IA* (Regulacja Obrazu), *Motion Detection MD* (Wykrywanie Ruchu), *Detection Estimation DE* (Ocena Detekcji) i *Background Update BU* (Aktualizacja Tła). W bloku IA poprzednio uaktualniony obraz tła, poprzednia klatka obrazu i aktualna klatka obrazu są przetwarzane. W efekcie czego system steruje ustawieniem kamery. Blok MD wykrywa poruszające się piksele. Blok DE zajmuje się interpretacją wyników uzyskanych przez blok MD. Następnie w bloku BU uaktualniany jest obraz tła.

W pracy [79] opisano system detekcji ruchu, który ma za zadanie zmniejszenie liczby wypadków powodowanych przez nieuwagę kierowców. Taki system poprawiłby także komfort jazdy kierowców. Podstawowym zadaniem takiego systemu jest wykrywanie przeszkód na drodze samochodu i reagowanie w odpowiedni sposób.

Schafer, Wasmeier, Ratke, Foppe i Preuss w [80] opisali system monitorujący Wieżę Olimpijską w Monachium. Budowla ta ma 291 metrów wysokości i waży ok. 52 ton. System opisany w pracy [80] rejestruje ruch obiektu. Wieża odchyla się od pionu na skutek słońca i wiatru. System ma za zadanie obserwować o jaki kąt odchyla się wieża i alarmować w przypadku gdy odchylenie jest zbyt duże. Ma to zapobiec katastrofie jaka wydarzyłaby się w przypadku upadku Wieży Olimpijskiej. Wczesne wykrycie nieprawidłowości pozwoli ewakuować ludzi z niebezpiecznego terenu.



**Rysunek 5.** Wieża Olimpijska w Monachium.

Opisane wyżej przykłady zastosowania systemów wykrywania ruchu to bardzo znikoma część z wszystkich zastosowań takich systemów. Wyżej opisane zastosowania zostały uznane przez autora za godne uwagi.

### **3. Metody dopasowywania bloków**

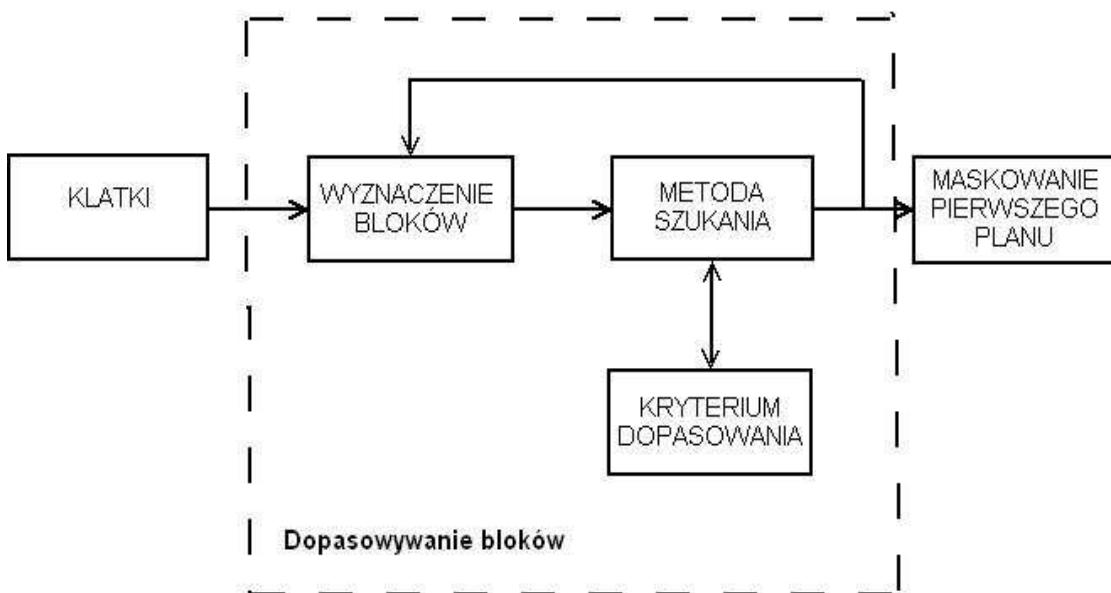
#### **3.1. Sieć wykrywania ruchu**

Jest kilka czy nawet kilkanaście algorytmów wykrywania ruchu przy pomocy metod porównywania bloków. Każdy algorytm składa się z kilku modułów (rysunek 6), które tworzą rodzaj sieci dopasowującej bloki. Różnice w realizacji tych modułów powodują rozbieżności w działaniu poszczególnych algorytmów. Lecz wspólnym celem wszystkich tych metod jest wykrycie poruszających się obiektów.

Zanim przedstawię szczegółowo każdy ze sposobów wykrywania ruchu tymi metodami chciałbym przybliżyć główne założenia działań poszczególnych modułów. Założenia te są dla wszystkich metod wspólne.

Ogólnie można stwierdzić, że metody dopasowywania bloków różnią się między sobą [1] :

- kształtem bloku
- metodą przeszukiwania klatki filmu
- kryterium dopasowania bloków (współczynnik dopasowania bloków)



**Rysunek 6.** Sieć dopasowująca bloki w algorytmach wykrywania ruchu

Na rysunku 6 widać ogólny schemat działania algorytmów wykrywających ruch za pomocą metod dopasowywania bloków. Jak widać te metody tworzą pewną sieć, w której poszczególne moduły realizują operacje niezbędne do wykrycia ruchu obiektu za pomocą tych metod [1]. Na wejście takiej sieci podawane są klatki filmu, w którym chcemy wykryć poruszające się obiekty. Po przetworzeniu takiej klatki przez moduły odpowiedzialne za dopasowywanie bloków na wyjściu otrzymujemy klatkę z zaznaczonymi przemieszczającymi się obiekty. Za zaznaczanie poruszających się obiektów odpowiada moduł *MASKOWANIE PIERWSZEGO PLANU*. Pozostałe moduły sieci dopasowującej bloki w algorytmach wykrywania ruchu:

- **WYZNACZENIE BLOKÓW** – ten moduł ustala rozmiar bloku oraz jego kształt. Określa także pozycję startową bloku z klatki odniesienia, którego nową pozycję będziemy ustalać na klatce aktualnie badanej.
- **METODA SZUKANIA** – ten moduł odpowiada za sposób przeszukiwania klatki badanej. Podczas przeszukiwania porównuje się dwa bloki (w klatce odniesienia i aktualnej).
- **KRYTERIUM DOPASOWANIA** – ten moduł ustala kryterium określające dopasowanie dwóch bloków. Zwracając wartość tego kryterium modułowi **METODA SZUKANIA** możliwe jest określenie czy znaleziono pozycję bloku z

klatki odniesienia w klatce aktualnie przetwarzanej. Parametrem określającym blok może być np. średnia intensywność jego pikseli.

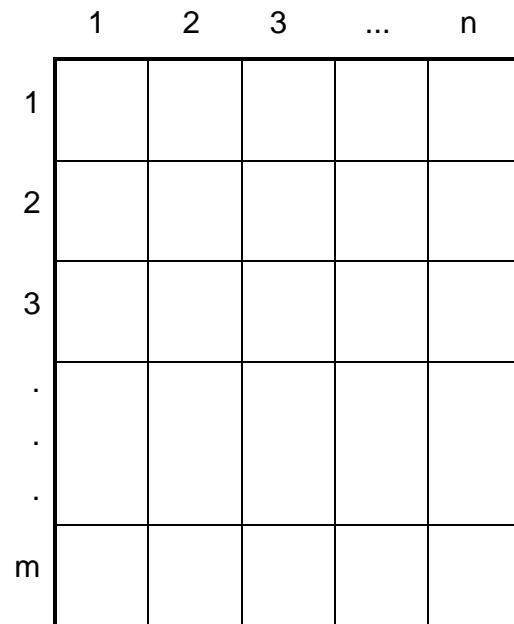
Ogólny schemat działania modułów odpowiedzialnych za dopasowywanie bloków jest następujący:

- podział klatki na bloki
- wybór odpowiedniego bloku (kandydata)
- sprawdzenie podobieństwa bloku-kandydata (klatka badana) do bloku-wzorca (w klatce odniesienia)
- jeśli znaleziono zadowalająco podobny blok to zaznacza się go na klatce wyjściowej (tej, na której zaznaczamy poruszające się obiekty). Jeśli nie to wybieramy kolejnego kandydata, a w przypadku ich braku stwierdzamy brak ruchu.

Algorytmy wykrywania ruchu obiektów za pomocą metod blokowych nadają się do działania w czasie rzeczywistym. Metody te wprowadzają pewne uproszczenia (dopasowujemy bloki – nie pojedyncze piksele) przez co ich złożoność obliczeniowa jest mniejsza. Lecz te uproszczenia mogą powodować pewne niepożądane działania algorytmów np. nie wykrycie ruchu „niewielkich” obiektów.

### 3.2. Podział klatki na bloki

Klatkę dzieli się na prostokątno-podobne bloki. Wszystkie bloki są tego samego rozmiaru [1][2][4][5]. W większości prac zaleca się stosowanie nie nachodzących na siebie bloków kwadratowych (o rozmiarze 4x4, 8x8, 16x16 pikseli itp.). Lecz można także stosować podział klatki na bloki o innym kształcie niż prostokątnym [6] np. trójkątnym.



**Rysunek 7.** Podział klatki na kwadratowe bloki

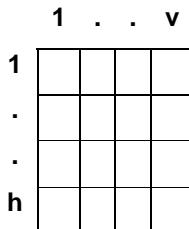
Jak widać na rysunku 7 klatkę dzielimy na  $m \times n$  bloków. Wprowadzamy blok jako:

$$B(i, j) \quad (12)$$

gdzie:  $i$  – pozycja pionowa bloku w klatce,  $i \in \langle 0, m \rangle$

$j$  – pozycja pozioma bloku w klatce,  $j \in \langle 0, n \rangle$

Blok składa się z  $h_v v$  pikseli. Gdzie  $h$  oznacza wysokość bloku a  $v$  jego szerokość (patrz rysunek 8). Rozmiar bloku ma dość duże znaczenie [2]. Im większy rozmiar bloku tym mniejszy wpływ zakłóceń. Natomiast wraz ze wzrostem rozmiaru bloku należy liczyć się ze zmniejszeniem „czułości ruchowej” algorytmu. Mianowicie chodzi o to, że obiekt może się poruszać w granicach bloku. Jeśli blok ma duży rozmiar algorytm nie wykryje ruchu. Można powiedzieć, że rozmiar bloku jest kompromisem pomiędzy odpornością algorytmu na szумy a wielkością obiektów, których ruch wykrywamy. Mniejszy blok pozwala na lepsze odwzorowanie krawędzi.



Rysunek 8. Budowa bloku o rozmiarze  $h \times v$

Jak widać rozmiar bloku ma bardzo duże znaczenie. Złe lub lekkomyślne jego dobranie może spowodować błędne działanie algorytmu. Parametr ten należy dobrać w zależności od obiektów, tła i przewidywanych zakłóceń.

### 3.3. Współczynniki dopasowania bloków

Kryterium dopasowania bloków to nic innego jak pewien współczynnik. Współczynnik taki określa w jakim stopniu blok z klatki aktualnie badanej jest podobny do bloku znajdującego się w klatce odniesienia. Biorąc pod uwagę pewne zakłócenia (np. niepożądane ruchy kamery, kolor tła podobny do koloru obiektu itp.) [2] nie należy się spodziewać idealnego dopasowania. Można stwierdzić, że im akceptowalna wartość współczynnika dopasowania będzie mniejsza tym mniej będzie nieprawidłowo wykrytych obiektów. Lecz może się zdarzyć, że algorytm nie będzie wykrywał poruszających się obiektów z powodu zbyt drastycznego określenia akceptowalnej wartości współczynnika. Ta akceptowalna wartość często jest nazywana **progiem T** [1] [2] [4]. Im próg  $T$  większy tym algorytm jest bardziej odporny na zakłócenia ale też więcej źle wykrytych podobnych bloków. Lecz niektóre algorytmy pozbawione są takiego progu.

Wprowadźmy pewne oznaczenia:

- $v_B$  – współczynnik dopasowania
- $I_c$  – aktualnie badana klatka
- $I_r$  – klatka odniesienia
- $p$  – pionowe przesunięcie bloku (z klatki badanej) w stosunku do bloku z klatki odniesienia

- **q** – poziome przesunięcie bloku (z klatki badanej) w stosunku do bloku z klatki odniesienia

W literaturze opisano kilka współczynników dopasowania dwóch bloków[1] [2] [3] [4] [5] [6]. W konkretnym algorytmie używamy jednego. Więc należy dokładnie zastanowić się, który współczynnik będzie najodpowiedniejszy dla naszego przypadku.

Współczynniki dopasowania [1]:

- **SAD** (*ang. Sum of the Absolute values of the Differences*) – współczynnik ten określa sumę wartości bezwzględnych różnic pomiędzy wartościami pikseli w dwóch blokach. Sprawdza się różnice w wartościach odpowiadających sobie pikseli w dwóch blokach (w klatce odniesienia i aktualnie badanej).

$$v_B = \sum_{i=1}^h \sum_{j=1}^v |I_c(i, j) - I_r(i + p, j + q)| \quad (13)$$

- **MAD** (*ang. Mean of the Absolute values of the Differences*) – podobnie jak SAD też liczy różnice w wartościach pikseli. Lecz tym razem liczona jest wartość średnia. Jak łatwo zauważyc jest to średnia arytmetyczna z SAD.

$$v_B = \frac{1}{h \cdot v} \sum_{i=1}^h \sum_{j=1}^v |I_c(i, j) - I_r(i + p, j + q)| \quad (14)$$

- **MSD** (*ang. Mean of the Square of the Differences*) – jest to średnia arytmetyczna z sumy kwadratów różnic pomiędzy blokami.

$$v_B = \frac{1}{h \cdot v} \sum_{i=1}^h \sum_{j=1}^v (I_c(i, j) - I_r(i + p, j + q))^2 \quad (15)$$

- **SSD** (ang. sum of squared difference) – współczynnik ten wyraża sumę kwadratów różnic pomiędzy pikselami dwóch porównywanych bloków [72].

$$v_B = \sum_{i=1}^h \sum_{j=1}^v (I_c(i, j) - I_r(i + p, j + q))^2 \quad (16)$$

- **NMP** (ang. Non-Matching Pixels) – współczynnik ten określa sumę nie pasujących pikseli w dwóch blokach. Określenie czy dwa piksele pasują do siebie określa pewien próg  $t_{NMP}$ . Liczy się różnicę wartości odpowiadających sobie pikseli w dwóch blokach. Następnie sprawdza się czy ta różnica jest mniejsza bądź równa ustalonego progu  $t_{NMP}$ .

$$v_B = \sum_{i=1}^h \sum_{j=1}^v D(I_c(i, j), I_r(i + p, j + q)) \quad (17)$$

$$D(a, b) = \begin{cases} 0 \Leftrightarrow |a - b| \leq t_{NMP} \\ 1 \Leftrightarrow |a - b| > t_{NMP} \end{cases} \quad (18)$$

- **NNMP** (ang. Number of Not Matching Points) – nie jest to typowy współczynnik dopasowania. Znajduje on zastosowania w algorytmach typu BBMA (ang. Binary Block Matching Algorithm) [42]. Współczynnik ten porównuje piksele z obrazu binarnego uzyskanego transformataj 1BT (ang. one-bit transform). W tych algorytmach jest on używany zamiast pozostałych współczynników jak np. SAD. Jest definiowany następująco:

$$NNMP(x, y) = \sum_{i=1}^h \sum_{j=1}^v B_i(i, j) \otimes B_{i-1}(i + x, j + y) \quad (19)$$

gdzie:  $B_i(i, j)$  jest wartością binarną piksela  $(i, j)$  z bloku  $B_i$  z obrazu binarnego z aktualnej klatki a  $B_{i-1}$  jest wartością binarną piksela  $(i+x, j+y)$  z bloku  $B_{i-1}$  z

obrazu binarnego z poprzedniej klatki przesuniętego o  $(x, y)$ . Natomiast  $\otimes$  oznacza operator logiczny Ex-OR (*ang. exclusive-or*).

- **MMAD** (*ang. Modified MAD*) – zmodyfikowany współczynnik MAD [64] [65]. Współczynnik jest definiowany następująco:

$$v_b = \frac{d}{h \cdot v} \cdot DMV(i, j) + \frac{1}{h \cdot v} \sum_{i=1}^h \sum_{j=1}^v |I_c(i, j) - I_r(i + p, j + q)| \quad (20)$$

gdzie:

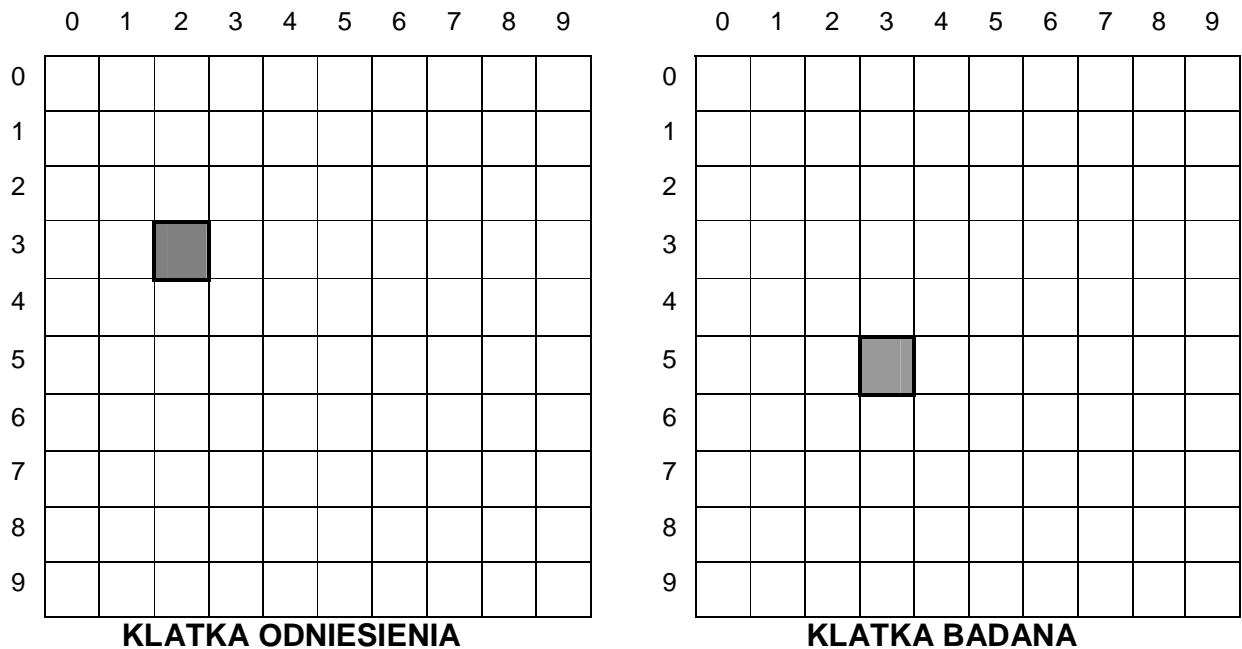
**d** – stała (zalecana wartość 1,5)

**DMV(i, j)** – długość wektora ruchu

Jak widać jest kilka możliwych do zastosowania współczynników. Nie można stwierdzić, że któryś jest najlepszy bądź najgorszy. Musimy sami zdecydować o tym, który będzie odpowiedni do „środowiska”, w którym wykrywamy poruszające się obiekty.

### 3.4. Wektor ruchu

Wektor ruchu (*ang. motion vector*) wskazuje na jaką pozycję w stosunku do klatki poprzedniej przesunął się blok z aktualnej klatki (patrz rysunek 9).



Rysunek 9. Ilustracja przedstawiająca wektor ruchu.

Na rysunku 9 widać, że blok (zaciemiony na szaro) znajdujący się na klatce badanej na pozycji (5, 3) znajdował się na klatce odniesienia na pozycji (3, 2). Widać więc, że blok przesunął się, czyli jest w ruchu. Tak więc wektor ruchu dla bloku z pozycji (3, 2) wynosi (5, 3). Tak więc wektor ruchu możemy zdefiniować następująco:

$$\vec{m}(i, j) = (i + p, j + q) \quad (21)$$

gdzie:

**(i, j)** – pozycja bloku na klatce odniesienia

**(i+p, j+q)** – pozycja bloku na klatce badanej

**p** – przesunięcie bloku w pionie

**q** – przesunięcie bloku w poziomie

Wektor ruchu  $\vec{m}(i, j) = (i, j)$  mówi o tym, że blok z pozycji  $(i, j)$  jest nieruchomy.

### 3.5. Algorytmy przeszukiwania

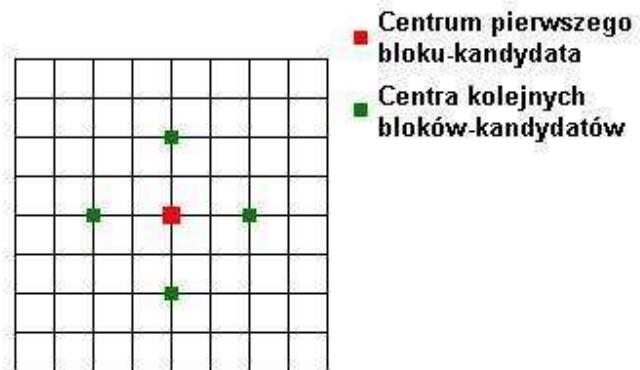
Algorytmy dopasowania bloków można podzielić ze względu na algorytmy przeszukiwania [7] [8]. Chciałbym teraz przedstawić najpopularniejsze algorytmy przeszukiwania:

- LOGS (2-D logarithmic search) – przeszukiwanie logarytmiczne [9]
- TSS (three-step search) – przeszukiwanie w trzech krokach [10]
- 4SS (four-step search) – przeszukiwanie w czterech krokach [11]
- BBGDS (block-based gradient descent search) – przeszukiwanie gradientowe [12]
- DS (diamond search) – przeszukiwanie diamentowe [13]
- HEXBS (Hexagon Based Search) – metoda sześciokąta [14]
- CS (Cross Search) – przeszukiwanie krzyżowe [13]
- NTSS (New TSS) – nowa metoda TSS [15]

Uważam, że można pozostać przy skrótach angielskich nazw. Więc w dalszej części pracy będę nazywał te przeszukiwania tymi skrótami np. LOGS.

Można stwierdzić, że algorytm przeszukiwania jest głównym i najważniejszym ogniwem algorytmu dopasowywania bloków. To te algorytmy decydują o tym, w którym kierunku będziemy szukać potencjalnie poruszających się bloków (obiektów). Oczywiście ważnym czynnikiem jest także dobre wybranie współczynnika dopasowania dwóch bloków i najodpowiedniejszego progu.

Wprowadźmy pojęcie **kroku S** (ang. step size), który będzie określał odległość od centrum bloku-kandydata do innych bloków kandydatów (rysunek 10) [1].



Rysunek 10. Obszar przeszukiwania wynoszące 9x9 bloków i krok wynoszący 2 bloki.

**Obszar przeszukiwania** (*ang. search window*) jest to obszar, w którym przesuwamy bloki w celu ich porównania z blokiem wzorcowym [1] [8] [7]. Wielkość obszaru zależy od kroku S.

Poniżej znajdują się opisy poszczególnych metod przeszukiwania klatek filmowych w celu wykrycia poruszających się algorytmów.

### 3.5.1. Metoda TSS

Metoda ta działa rekurencyjnie. Spośród ośmiu pozycji sąsiednich bloków w każdym z trzech kroków wybiera jedną pozycję. Pozycja ta jest określana na podstawie najlepszego dopasowania dwóch bloków (blok wzorcowy z blokiem badanym). W każdym kroku algorytmu zmniejsza się krok S o połowę [1] [8].

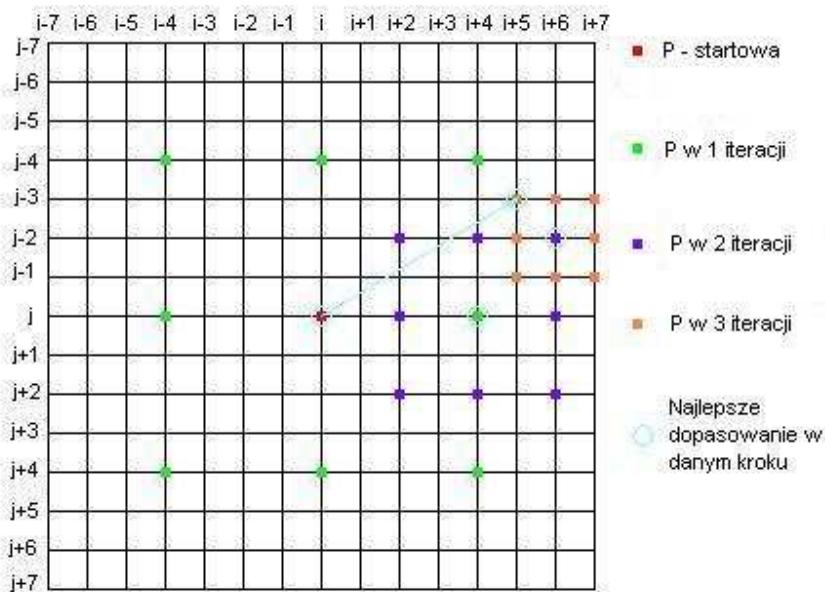
Metoda TSS redukuje liczbę bloków-kandydatów przy przeszukaniu dużego obszaru. Właściwość ta pozwala używać tej metody w technikach szybkiego przeszukiwania bloków. TSS obejmuje obszar o powierzchni wynoszącej  $(2^{(s+1)} - 1)^2$  z  $8(\log_2 s + 1) + 1$  blokami kandydującymi.

**Algorytm TSS:**

- Krok 1:** Ustaw aktualną pozycję centralną  $P = (i, j)$ . Policz dopasowanie bloku (bloku z pozycji  $P$  i bloku z pozycji  $(i, j)$  z klatki odniesienia)  $v_B$ . Jeśli  $v_B$  jest mniejsze od założonego progu  $T$  to przerwij szukanie. W innym przypadku krok 2.
- Krok 2:** Policz wartość dopasowania  $v_B$  w ośmiu pozycjach (8 sąsiadów) oddalonych o krok  $S$  od aktualnej pozycji centralnej  $P = (i, j)$ . Przesuń  $P$  na pozycję bloku, którego wartość  $v_B$  jest najmniejsza.
- Krok 3:** Jeśli  $S > 1$  to ustaw  $s = s/2$  i idź do kroku 2. W innym przypadku zwróć pozycję  $P$  jako wektor ruchu  $\vec{m}(i, j)$ .

Jak widać algorytm ten zwraca wektor ruchu  $\vec{m}(i, j)$ , który mówi o tym na jaką pozycję przesunął się blok. Mowa tu o bloku z klatki odniesienia i jego pozycji w klatce aktualnie badanej. Innymi słowy wektor ten mówi nam o tym, gdzie obecnie znajduje się poruszający się blok.

Jak można zauważyć ważną rolę odgrywa tutaj próg  $T$ . Jego wartość jest kluczowa dla nie poruszających się bloków. Dlatego, że blok który się nie porusza wcale nie musi mieć dopasowania bliskiego zeru. Zakłócenia i szумy mogą spowodować pewne odchylenia w wartościach pikseli. Przykład działania algorytmu przedstawiono na rysunku 11.



**Rysunek 11.** Przykład działania algorytmu TSS. Krok S = 4.

Teraz chciałbym omówić przykład z rysunku 11. Algorytm zaczyna działanie w pozycji  $P = (i, j)$ . Następnie policzono wartość  $v_b$ , która jest większa od progu T. Następnie liczymy wartości  $v_b$  w pozycjach sąsiednich oddalonych o krok S. Czyli w pozycjach  $(i, j-4)$ ,  $(i-4, j-4)$ ,  $(i-4, j)$ ,  $(i-4, j+4)$ ,  $(i, j+4)$ ,  $(i+4, j+4)$ ,  $(i+4, j)$  oraz  $(i+4, j-4)$ . Minimalna wartość  $v_b$  znajduje się w pozycji  $(i+4, j)$ . Algorytm przesuwa pozycję  $P$  na  $(i+4, j)$  oraz zmniejsza krok S na wartość S=2. Powtarzamy obliczenia  $v_b$  dla kolejnych pozycji oddalonych od  $P$  o krok S. Teraz minimalna wartość  $v_b$  znajduje się w pozycji  $(i+6, j-2)$ . Znowu zmniejszamy wartość kroku S. Tym razem S=1 i przeprowadzamy obliczenia  $v_b$  w ośmiu pozycjach. Znajdujemy minimalne  $v_b$  w pozycji  $(i+5, j-3)$ . Tym razem S nie jest większe od 1 więc algorytm kończy działanie i zwraca wektor ruchu  $\bar{m}(i, j) = (i + 5, j - 3)$ . Więc blok, który znajdował się na klatce odniesienia w pozycji  $(i, j)$  przemieścił się. W klatce obecnie badanej ten blok znajduje się w pozycji  $(i+5, j-3)$ . Innymi słowy wektor ruchu wynosi  $(i+5, j-3)$ .

Jak widać w metodzie TSS zaleca się obszar okna przeszukiwania obejmował 14x14 bloków [1].

### 3.5.2. Metoda 4SS

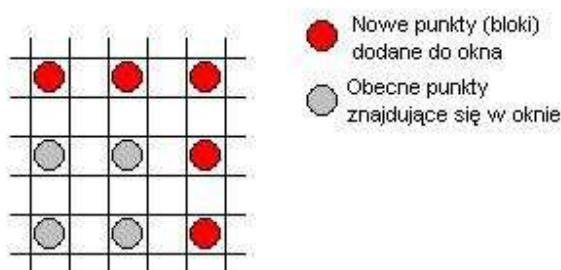
W tej metodzie zaproponowano aby w pierwszym kroku sprawdzać 9 punktów w oknie o rozmiarze wynoszącym 5x5. Przypominam, że w metodzie TSS obszar ten wynosił 9x9. Wielkość okna zależy od pozycji w jakiej znaleziono minimalne  $v_b$ , czyli dopasowanie dwóch bloków. W zależności od tej pozycji okno może wynosić 3x3 lub 5x5 [11].

#### Algorytm 4SS:

**Krok 1:** Liczymy dopasowanie bloku  $v_b$  dla 9 pozycji usytuowanych w oknie 5x5 ulokowanym w centrum obszaru przeszukiwań. Jeśli minimalna wartość  $v_b$  znajduje się w centrum okna to idź do kroku 4. W innym przypadku idź do kroku 2.

**Krok 2:** Zachowujemy rozmiar okna – 5x5. Szablon przeszukiwania zależy od pozycji poprzednio znalezionego minimalnego  $v_b$ . I tak:

- jeśli poprzednia pozycja punktu (bloku) o minimalnym  $v_b$  ulokowana była w rogu poprzedniego okna to 5 dodatkowych punktów dodajemy do obecnego okna według wzoru:



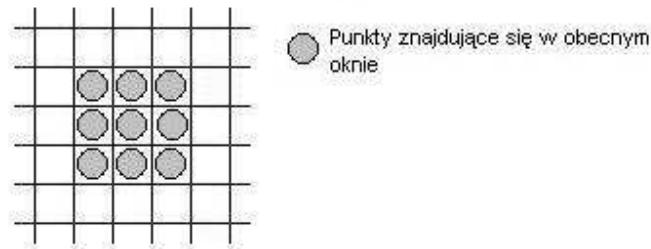
- jeśli zaś poprzednia pozycja punktu znajdowała się w środku poziomej lub pionowej krawędzi okna to 3 dodatkowe punkty dodajemy według wzoru:



W nowo utworzonym oknie liczymy wartości  $v_b$  dla 9 punktów. Jeśli punkt, dla którego minimalna wartość  $v_b$  znajduje się w centrum okna to idź do kroku 4. W przeciwnym wypadku idź do kroku 3.

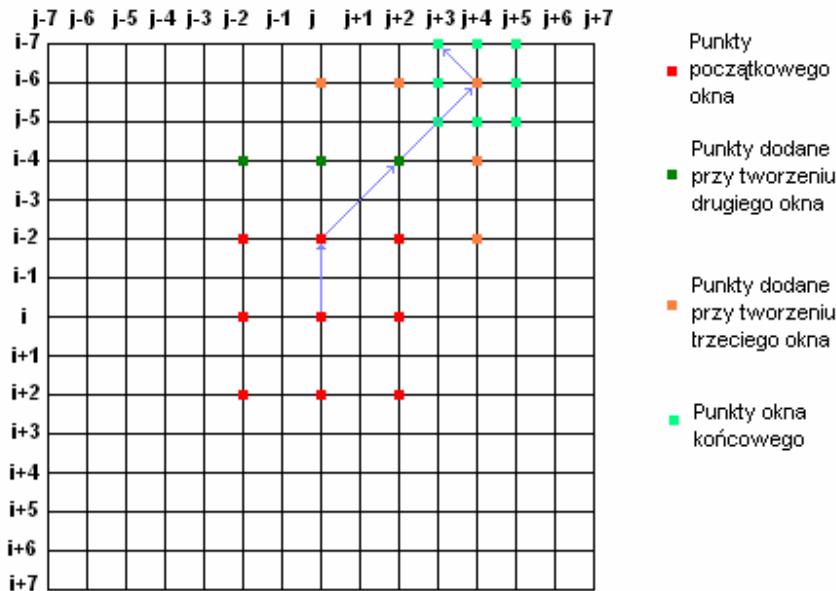
**Krok 3:** Model strategii przeszukiwania jest taki sam jak w kroku 2 z tą różnicą, że na koniec przechodzi się do kroku 4.

**Krok 4:** Okno mające rozmiar 3x3 ma następujący wygląd:



Liczmy wartości  $v_b$  w 9 punktach (blokach). Minimalna wartość  $v_b$  jest pozycją, w której znajduje się na aktualnej klatce blok z klatki odniesienia. Innymi słowy to jest wektor ruchu badanego bloku.

Przykład działania algorytmu 4SS znajduje się na rysunku 12. Jak można zauważyć algorytm ten wykonuje tylko cztery kroki (stąd też jego nazwa). W kolejnych krokach punkty „nakładają się” na siebie. Teoretycznie powinno to dawać lepsze określenie nowej pozycji bloku. Dzieje się tak dlatego, że nie traci się najlepszej wartości dopasowania z poprzedniego kroku. W metodzie TSS takie ryzyko występuje.



Rysunek 12. Przykład działania algorytmu 4SS

W przykładzie z rysunku 12 pokazano jak działa algorytm 4SS. Najpierw policzono wartości  $v_b$  dla 9 punktów (bloków). Najlepszą wartością okazała się ta dla punktu z pozycji  $(i-2, j)$ . Jak widać punkt ten leży w środku poziomej krawędzi naszego okna. Zatem dodajemy nowe punkty (zielone) i otrzymujemy nowe okno. Następnie liczymy wartości  $v_b$  dla punktów nowego okna. Z 9 policzonych wartości  $v_b$  najlepsza okazuje się ta, którą posiada punkt z pozycji  $(i-4, j+2)$ . Tym razem punkt ten leży w rogu naszego okna. Zatem dodajemy nowe punkty (pomarańczowe). Tym razem najlepszy punkt znajduje się w pozycji  $(i-6, j+4)$ . W dotychczasowym działaniu algorytmu okno miało rozmiar  $5 \times 5$  lecz w tym kroku musimy zmienić ten rozmiar na  $3 \times 3$ . Dodajemy nowe punkty do okna (jasno-zielone). Po wykonaniu obliczeń okazuje się, że najlepszym punktem jest ten z pozycji  $(i-7, j+3)$ . Zatem blok z klatki odniesienia przesunął się na aktualnie badanej klatce. Wektor ruchu tego bloku wynosi  $\vec{m}(i, j) = (i - 7, j + 3)$ , czyli badany blok się porusza.

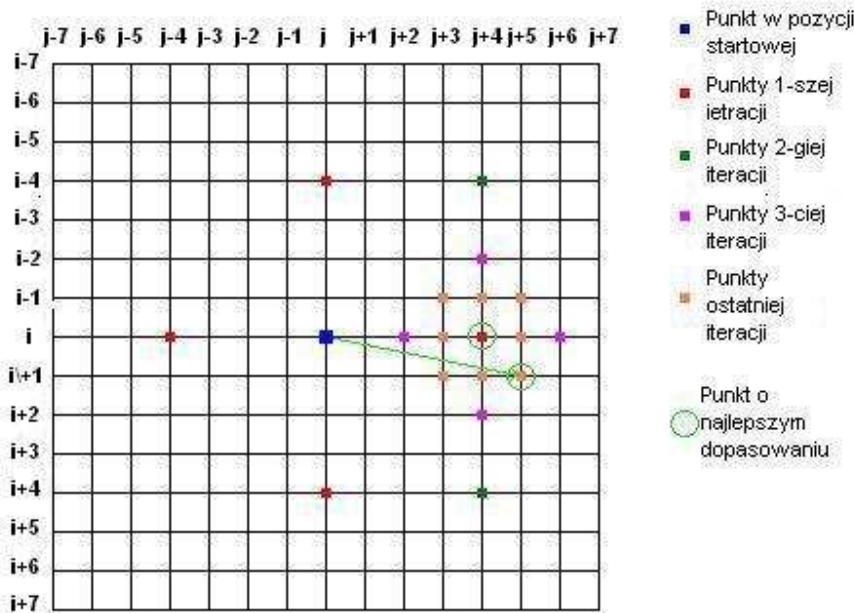
Ten algorytm można stosować w systemach wykrywania ruchu obiektów, które działają w czasie rzeczywistym [11]. W najgorszym przypadku będziemy sprawdzać dopasowania dla 27 bloków (punktów).

### 3.5.3. Metoda LOGS

Pełna nazwa tego algorytmu to przeszukiwanie logarytmiczne (*ang. 2D-logarithmic search*). To przeszukiwanie może być szybsze niż to metodą TSS [1] [9]. W algorytmie tym w stosuje się różną liczbę przeszukiwanych punktów (bloków). Podobnie jak w TSS także w tej metodzie następuje zmniejszanie kroku S. Okno punktów do przeszukania ma ich 5 lub 9. W zależności od kroku, w którym znajduje się algorytm.

#### **Algorytm LOGS:**

- Krok 1:** Policz dopasowanie  $v_b$  dla bloku z pozycji  $(i, j)$ . Ustaw centrum okna  $P = (i, j)$ . Jeśli otrzymana wartość  $v_b$  jest mniejsza od zakładanego progu  $T$  to zakończ przeszukiwanie. W innym przypadku idź do kroku 2.
- Krok 2:** Policz wartości  $v_b$  dla czterech bloków oddalonych od  $P$  o krok S (tworzących krzyż). Ustaw  $P$  na pozycji bloku o minimalnym  $v_b$ . Jeśli  $P$  nie znajduje się w centralnym punkcie okna to idź do kroku 2. W innym przypadku idź do kroku 3.
- Krok 3:** Ustaw  $S=S/2$ . Jeśli  $S > 1$  to idź do kroku 2. W innym przypadku idź do kroku 4.
- Krok 4:** Policz dopasowania  $v_b$  dla ośmiu bloków oddalonych od  $P$  o krok S (czyli o 1). Punkt o najlepszym dopasowaniu  $v_b$  jest wektorem ruchu  $\vec{m}(i, j)$  dla bloku  $(i, j)$



Rysunek 13. Przykład działania algorytmu LOGS.

Na powyższym rysunku 13 znajduje się przykład działania algorytmu LOGS. Krok S=4. Działanie algorytmu zaczyna się w punkcie  $(i, j)$ . Liczymy wartość  $v_b$  dla bloku znajdującego się w tym punkcie. Następnie liczymy wartości  $v_b$  dla bloków znajdujących się na pozycjach  $(i, j-4)$ ,  $(i-4, j)$ ,  $(i+4, j)$ ,  $(i, j+4)$ . Czyli liczymy wartości  $v_b$  dla czterech punktów oddalonych od centralnego punktu okna  $P$ , znajdującego się w  $(i, j)$ , oddalonych od niego o krok S. Najlepszym punktem okazuje się ten z pozycji  $(i, j+4)$ . Punkt ten nie znajduje się w centrum naszego okna więc krok S pozostaje bez zmian. W następnej iteracji pojawiają się nowe punkty (zielone) oddalone o  $P$  (znajdującym się w  $(i, j+4)$ ) o krok S. Tym razem liczymy wartości dla wszystkich punktów naszego okna i okazuje się, że punkt  $(i, j+4)$  znów jest najlepszy. Więc P pozostaje bez zmian ale S zmniejszamy o połowę. Po utworzeniu nowego okna (punkty fioletowe) liczymy wartości  $v_b$  dla wszystkich jego punktów. Znów punkt  $(i, j+4)$  okazuje się najlepszy. Lecz tym razem S jest równe 1 (po zmniejszeniu). Więc w nowym oknie znajdzie się więcej bloków (będzie ich 9). Dla nowego okna (punkty pomarańczowe) liczymy wartości  $v_b$ . Tym razem najmniejszą wartość  $v_b$  ma blok znajdujący się w pozycji  $(i+1, j+5)$ . Więc wektor ruchu dla bloku  $(i, j)$  wynosi

$\vec{m}(i, j) = (i+1, j+5)$ . Innymi słowy blok, który w klatce odniesienia znajdował się w pozycji  $(i, j)$  w aktualnie przetwarzanej klatce przesunął się na pozycję  $(i+1, j+5)$ .

### 3.5.4. Metoda CS

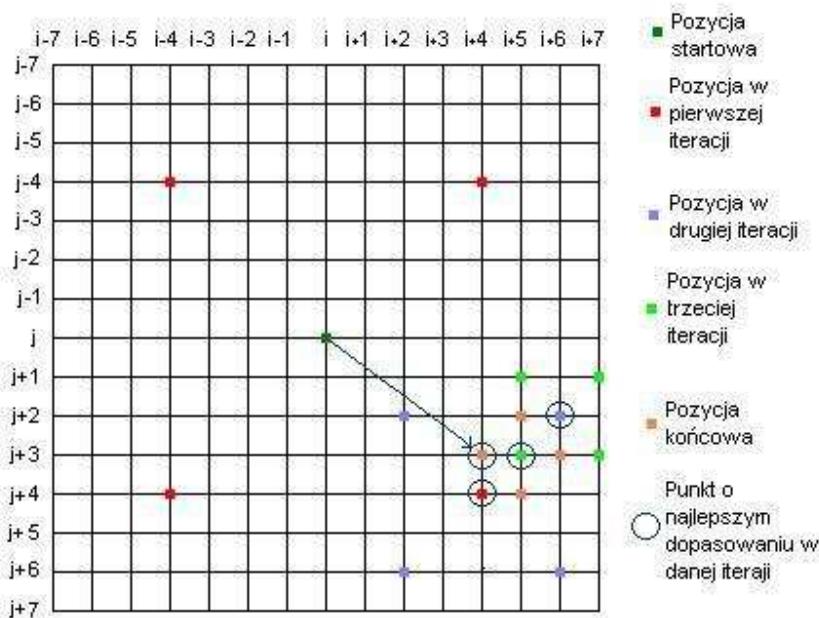
Metoda przeszukiwania krzyżowego CS (*ang. cross search*) jest podobna do TSS. Z tą różnicą, że tu blok położony w centralnej pozycji  $P$  posiada czterech sąsiadów. Sąsiedzi ci tworzą **X** lub **+** w zależności od kroku algorytmu [1] [13]. Przeszukiwanie jest szybsze niż metodą TSS ze względu na mniejszą liczbę sprawdzanych bloków.

#### Algorytm CS:

- Krok 1:** Dla bloku z pozycji  $(i, j)$  policz dopasowanie  $v_b$ . Jeśli wartość  $v_b$  jest mniejsza od założonego progu  $T$  to zakończ działanie algorytmu. W innym przypadku ustaw pozycję centralną  $P = (i, j)$  i idź do kroku 2.
- Krok 2:** Policz dopasowania  $v_b$  dla czterech sąsiadów oddalonych od bloku z pozycji  $P$  o krok  $S$ . Sąsiedzi tworzą **X** z blokiem z pozycji  $P$  w środku. Ustaw  $P$  na pozycji bloku o minimalnym  $v_b$ .
- Krok 3:** Ustaw  $S = S/2$ . Jeśli  $S > 1$  to idź do kroku 2. W innym przypadku idź do kroku 4.
- Krok 4:** Jeśli  $P$  znajduje się w prawym lub lewym dolnym rogu krzyża (**X**) to idź do kroku 5. W innym przypadku idź do kroku 6.
- Krok 5:** Policz dopasowania  $v_b$  dla czterech sąsiadów bloku z pozycji  $P$  oddalonych od niego o  $S$ . Sąsiedzi tworzą **+** z blokiem z pozycji  $P$  w środku. Pozycja bloku o minimalnej wartości  $v_b$  jest wartością wektora ruchu  $\vec{m}(i, j)$ .

**Krok 6:** Policz dopasowania  $v_b$  dla czterech sąsiadów bloku z pozycji  $P$  oddalonych od niego o 1. Sąsiedzi tworzą **X** z blokiem z pozycji  $P$  w środku. Pozycja bloku o minimalnej wartości  $v_b$  jest wartością wektora ruchu  $\vec{m}(i, j)$ .

Jak widać metoda ta jest bardzo podobna do metody TSS. W metodzie CS jest mniej bloków-kandydatów niż w TSS. Przykład działania metody CS ilustruje rysunek 14.



Rysunek 14. Przykład działania algorytmu CS (przeszukiwania krzyżowego).

W przykładzie z rysunku 14 krok  $S$  początkowo wynosi 4. Algorytm zaczyna swoje działanie w pozycji  $(i, j)$ . Dla bloku z pozycji  $(i, j)$  uzyskana wartość  $v_b$  jest większa od progu  $T$ . Zatem algorytm ustawia pozycję centralną  $P = (i, j)$ . Następnie liczy wartości dopasowania  $v_b$  dla sąsiadów tworzących krzyż **X** (punkty czerwone). Najmniejszą wartość  $v_b$  posiada blok z pozycji  $(i+4, j+4)$ . Więc algorytm zmniejsza krok  $S$  o połowę i znów liczy wartości dopasowania bloków sąsiednich (punkty fioletowe). Tym razem najlepsze dopasowanie posiada blok z pozycji  $(i+6, j+2)$ . Więc algorytm ustawia pozycję centralną  $P=(i+6, j+2)$ . Znów algorytm liczy dopasowania  $v_b$  dla

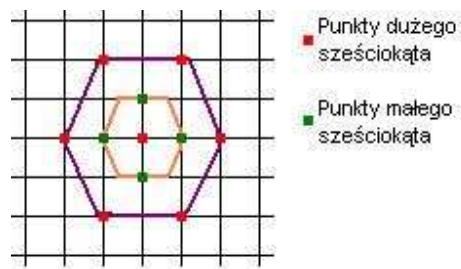
sąsiadów (punkty jasno-zielone) bloku z pozycji  $P$ . Tym razem najlepsze dopasowanie posiada blok z pozycji  $(i+5, j+3)$ . Po zmniejszeniu kroku S okazuje się, że wynosi on 1. Zatem przechodzimy do ostatniego kroku. Ostatni znaleziony punkt znajdował się w dolnym lewym rogu krzyża. To powoduje, że teraz sąsiedzi tworzą krzyż + z punktem  $P=(i+5, j+3)$  w środku. Po obliczeniu dopasowania  $v_b$  dla każdego z punktów znajdujemy blok o najlepszym dopasowaniu na pozycji  $(i+4, j+3)$ . Więc blok z pozycji  $(i, j)$  z klatki odniesienia znajduje się w klatce aktualnie badanej na pozycji  $(i+4, j+3)$ . Innymi słowy wektor ruchu dla bloku  $(i, j)$  wynosi  $\vec{m}(i, j) = (i+4, j+3)$ .

### 3.5.5. Metoda HEXBS

Metoda sześciokąta HEXBS (ang. *Hexagon-Based Search*) wykorzystuje do przeszukiwania wzór sześciokąta [14]. Przy przeszukiwaniu wykorzystuje się sześciokąty (rysunek 8) dwóch rodzajów:

- dużego
- małego

Należy zwrócić uwagę na to, że sześciokąt mały tworzą cztery bloki. Natomiast duży sześciokąt tworzy sześć bloków. Zatem algorytm ten na starcie liczy dopasowania  $v_b$  dla sześciu bloków.



Rysunek 15. Sześciokąty wykorzystywane w metodzie HEXBS

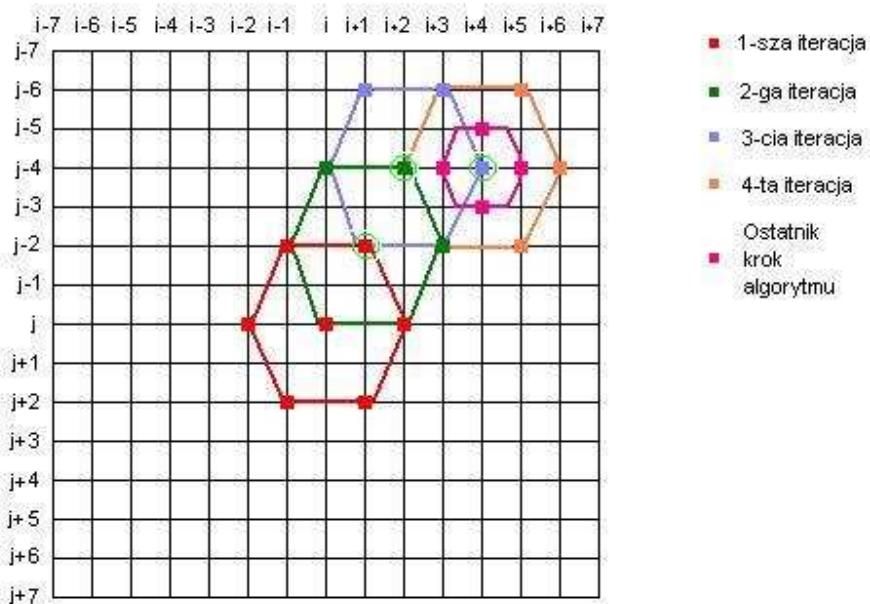
#### Algorytm HEXBS:

- Krok 1:** Policz dopasowanie  $v_b$  dla bloku z pozycji  $(i, j)$ . Jeśli  $v_b$  jest mniejsze od założonego progu  $T$  to zakończ działanie algorytmu i ustaw wektor ruchu  $\vec{m}(i, j) = (i, j)$ . W przeciwnym wypadku ustaw  $P = (i, j)$  i idź do kroku 2.
- Krok 2:** Policz dopasowania  $v_b$  dla 7 bloków tworzących duży sześciokąt o środku w  $P$ . Ustaw  $P$  na pozycji bloku o najmniejszej wartości  $v_b$ . Jeśli  $P$  znajduje się w centrum sześciokąta to idź do kroku 3. W innym przypadku idź do kroku 2.
- Krok 3:** Policz dopasowania  $v_b$  dla 4 bloków tworzących mały sześciokąt o środku w  $P$ . Pozycja bloku o najmniejszej wartości  $v_b$  jest wartością wektora ruchu  $\vec{m}(i, j)$  dla bloku z pozycji  $(i, j)$ . Zakończ działanie algorytmu.

Jak można zauważyć w algorytmie tym należy ustalić wartość kroku S. Na tej podstawie zostanie utworzony duży sześciokąt o boku dwa razy większym od boku małego sześciokąta. Przykład działania tego algorytmu znajduje się na rysunku 9. Ilość bloków  $N$ , dla których algorytm obliczy wartość dopasowania  $v_b$  można obliczyć ze wzoru:

$$N = 7 + 3 \cdot n + 4 \quad (22)$$

gdzie **n** jest ilością wykonania kroku 2 przez algorytm.



**Rysunek 16.** Przykład działania algorytmu HEXBS

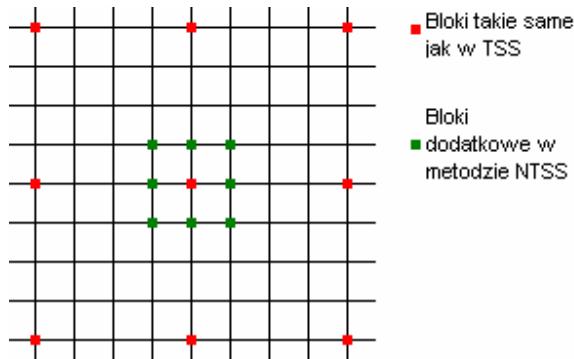
Na powyższym rysunku znajduje się przykład działania algorytmu HEXBS. Działanie algorytmu zakończyło się w 5 iteracjach. W pierwszej iteracji najlepsze dopasowanie posiadał blok na pozycji  $(i+1, j-2)$ . W drugiej najlepsze dopasowanie miał ten z pozycji  $(i+2, j-4)$ . W trzeciej iteracji ten z pozycji  $(i+4, j-4)$ . W następnej iteracji znów ten blok miał najlepsze dopasowanie. Lecz tym razem blok ten znajdował się w centrum dużego sześciokąta. Zatem w następnej iteracji blok ten był środkiem małego sześciokąta. Spośród bloków należących do małego sześciokąta znów blok z pozycji  $(i+4, j-4)$  miał najlepsze dopasowanie. Zatem wektor ruchu bloku z pozycji  $(i, j)$  ma wartość  $\vec{m}(i, j) = (i + 4, j - 4)$ . Oznacza to, że blok z pozycji  $(i, j)$  z klatki odniesienia na klatce aktualnie przetwarzanej znajduje się na pozycji  $(i+4, j-4)$ . W tym przykładzie widać, że w każdym kolejnym kroku dokłada się trzy punkty i w ten sposób tworzy się nowe duże sześciokąty. Ten proces powtarza się dopóki w centralnym punkcie nie znajdzie się blok o najlepszym dopasowaniu. Wtedy to zamiast dużego tworzymy mały sześciokąt.

### 3.5.6. Metoda NTSS

Metoda NTSS (*ang. New Three-Step Search*) jest metodą, która opiera się na metodzie TSS [15]. Różnica pomiędzy tymi metodami polega na dodaniu kilku kroków w metodzie NTSS. W metodzie NTSS w stosunku do TSS wyeliminowano porównanie wartości dopasowania z progiem ustalonym przez użytkownika.

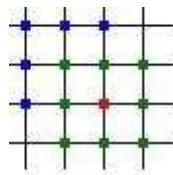
#### **Algorytm NTSS:**

**Krok 1:** Centralny punkt  $P$  znajduje się na pozycji bloku  $(i, j)$ . Policz wartości dopasowania  $v_b$  dla 17 bloków. Te bloki to: blok położony w pozycji  $P$ , 8 bloków oddalonych od  $P$  o krok  $S$  (czyli tak samo jak w TSS) oraz dodatkowe bloki wokół punktu  $P$  (tak jak na poniższym rysunku).

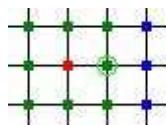


Jeśli minimalną wartość posiada blok znajdujący się na pozycji  $P$  to zakończ szukanie. Jeśli minimalną wartość posiada blok będący sąsiadem bloku z pozycji  $P$  to idź do kroku 2. W innym przypadku postępuj tak jak w metodzie TSS.

**Krok 2:** Dodaj nowe bloki w zależności od pozycji bloku o minimalnym  $v_b$ . Jeśli taki blok znajduje się w jednym z rogów kwadratu, który tworzą bloki sąsiednie punktu  $P$  to nowe bloki znajdują się w pozycjach jak na poniższym rysunku:



Jeśli natomiast taki blok znajduje się w innym miejscu to nowe bloki dodajemy tak:

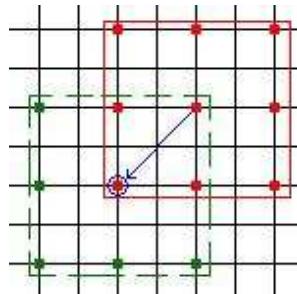


Następnie liczymy wartości dopasowania  $v_b$  dla nowych bloków. Pozycja bloku, dla którego  $v_b$  jest najmniejsze jest wartością wektora ruchu bloku  $(i, j)$ .

Wydaje się, że algorytm ten może zostać zastosowany w systemach, gdzie spodziewamy się obiektów wolno poruszających się. Algorytm ten tak naprawdę składa się z algorytmu TSS i dwóch dodatkowych kroków. Te dwa kroki decydują o tym, że ten algorytm szybciej znajdzie blok, który przesunął się w niewielkim stopniu. Można powiedzieć, że jest to ulepszony algorytm TSS [15].

### 3.5.7. Metoda BBGDS

Metoda BBGDS (*ang. Block-Based Gradient Descent Search*) używa kwadratu zbudowanego z 3x3 bloków. Kwadrat ten porusza się w obszarze przeszukiwania. W tej metodzie krok S jest stały [12] [16]. Na rysunku 10 widać jak porusza się ten kwadrat. To na jaką pozycję się przesunie zależy od pozycji bloku o najlepszym dopasowaniu.



Rysunek 17. Sposób poruszania się kwadratu zbudowanego z bloków

W dalszej części pracy kwadratem przeszukującym będę określał kwadrat zbudowany z bloków o rozmiarze  $3 \times 3$  bloki.

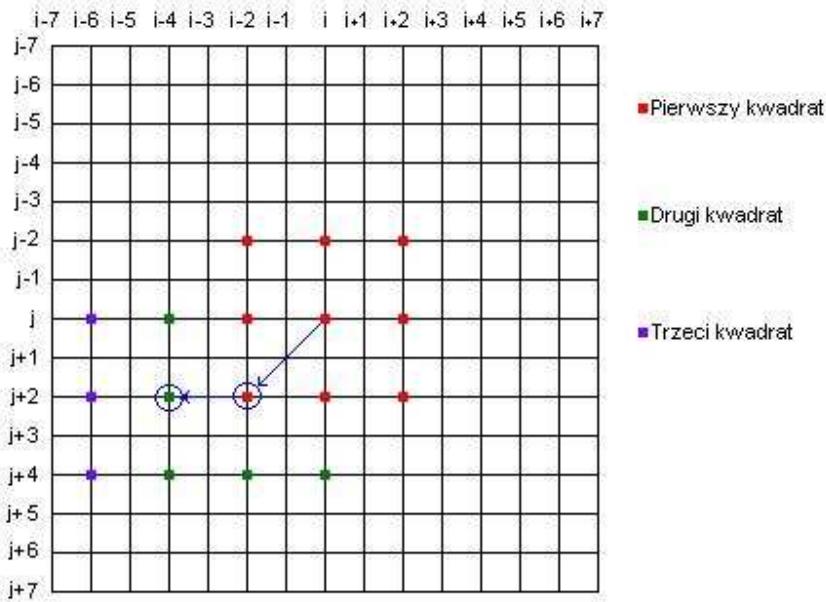
Jak widać na rysunku 10 kwadrat ten nie zmienia swojego rozmiaru. Jego centralny punkt (blok) przemieszcza się na pozycje bloku o najlepszym dopasowaniu.

#### Algorytm BBGDS:

**Krok 1:** Centralnym blokiem kwadratu jest blok na pozycji  $(i, j)$ . Policz dopasowania  $v_b$  dla 9 bloków wchodzących w skład kwadratu przeszukującego. Jeśli minimalną wartość  $v_b$  posiada blok z pozycji centralnej P to idź do kroku 3. W innym przypadku ustaw pozycję centralną P na pozycji bloku o minimalnym  $v_b$  i idź do kroku 2.

**Krok 2:** Policz wartości dopasowania  $v_b$  dla 9 bloków kwadratu przeszukującego o środku w pozycji P. Jeśli minimalną wartość  $v_b$  osiąga blok z pozycji P to idź do kroku 3. W innym przypadku ustaw P na pozycji bloku o minimalnym  $v_b$  i idź do kroku 2.

**Krok 3:** Zakończ szukanie. Wartością wektora ruchu dla bloku  $(i, j)$  jest pozycja centralna P.



**Rysunek 18.** Przykład działania algorytmu BBGDS

Na rysunku 18 znajduje się przykład działania algorytmu BBGDS. W tym przykładzie krok S wynosił 2. Działanie algorytmu zaczyna się w pozycji  $(i, j)$ . Pozycja ta jest pozycją centralną. Po obliczeniu wartości dopasowania  $v_b$  bloków kwadratu przeszukiwania (kolor czerwony) okazało się, że minimalną wartość posiada blok z pozycji  $(i-2, j+2)$ . Więc budujemy nowy kwadrat (kolor zielony) tym razem jego środek jest na pozycji  $P=(i-2, j+2)$ . Blok o najlepszym dopasowaniu to blok na pozycji  $(i-4, j+2)$ . Więc  $P=(i-4, j+2)$  i dla tej pozycji budujemy nowy kwadrat (kolor fioletowy). Tym razem blok o najlepszym dopasowaniu to blok znajdujący się na pozycji  $P$ . Zatem algorytm kończy działanie i zwraca wektor ruchu dla bloku  $(i, j)$   $\bar{m}(i, j) = (i - 4, j + 2)$ .

### 3.5.8. Algorytm FBBMA

Algorytm FBBMA (ang. *Fast Binary Block Matching Algorithm*) zaproponowali w pracy [42] Ye-Kui Wang wraz z Guo-Fang Tu. Algorytmy BBMA (ang. *Binary Block Matching Algorithms*) opierają się na transformacji jednobitowej 1BT (ang. *one-bit*

*transform).* Z wersjami algorytmów BBMA innymi niż FBBMA można się spotkać w pracach [44][45]. Pierwszym krokiem algorytmów BBMA jest transformata obrazu z pełnej rozdzielczości (najczęściej 8-bitów/piksel) do rozdzielczości 1-bit/piksel. Jako współczynnik dopasowania stosowany jest współczynnik NNMP (*ang. Number Not Matching Points*).

Teraz chciałbym opisać sposoby transformacji 1BT. Koncepcję transformacji 1BT po raz pierwszy opisał Lee w [43]. Transformacja ta wykorzystuje średnią wartość szarości pikseli w bloku. Każdy 8-bitowy piksel z bloku jest przekształcany do postaci binarnej. Piksele w postaci binarnej tworzą obraz binarny (*ang. bit-plane*). Zależność według, której dokonywana jest transformacja pikseli 8-bitowych na piksele binarne przedstawia się następująco:

$$B(i, j) = \begin{cases} 1 \Leftrightarrow I(i, j) \geq bm \\ 0 \Leftrightarrow I(i, j) < bm \end{cases}$$

gdzie:  $B(i, j)$  jest wartością piksela z pozycji  $(i, j)$  na obrazie binarnym,  $I(i, j)$  jest wartością piksela z pozycji  $(i, j)$  na obrazie źródłowym,  $bm$  to średnia wartość pikseli w danym bloku na obrazie źródłowym. Powyższa wersja transformacji 1BT nazywana jest **metodą BMT** (*ang. block mean thresholding*).

W pracach [44][46] Feng zaproponował wykorzystanie transformaty 1BT jako kroku wstępnego dla klasycznych metod FS (*ang. fast search*). Ma to na celu wyeliminowanie mało prawdopodobnych lokacji z obrazu/klatki, a następnie dla pozostałych lokacji zastosować klasyczne metody FS.

Inną wersją transformacji 1BT jest **EDB** (*ang. edge-detection based*) i zaproponował ją Mizuki w [47]. W tej metodzie obraz binarny przedstawia mapę krawędziową obrazu źródłowego. Sposób transformacji jest następujący: piksel binarny przyjmuje wartość 1 gdy odpowiadający mu piksel z obrazu źródłowego jest piksem krawędziowym i 0 w przeciwnym wypadku. W pracach [47][48] wykazano, że metoda ta nie może być stosowana w blokach o małej liczbie pikseli krawędziowych. Metoda ta jest także szczególnie narażona na wszelkiego rodzaju szumy (np. dym, deszcz itp.).

Kolejną metodą transformacji jest **metoda FT** (*ang. filter transform*) przedstawiona w [45]. W metodzie tej każda 8-bitowa klatka  $I$  jest filtrowana jądrem  $K$  o rozmiarze  $17 \times 17$  w rezultacie czego powstaje klatka  $I'$ . Obraz binarny powstaje zgodnie z zależnością:

$$\begin{aligned} K(i, j) &= \begin{cases} 1/25 \Leftrightarrow i, j \in [0, 4, 8, 12, 16] \\ 0 \Leftrightarrow i, j \notin [0, 4, 8, 12, 16] \end{cases} \\ B(i, j) &= \begin{cases} 1 \Leftrightarrow I(i, j) \geq I'(i, j) \\ 0 \Leftrightarrow I(i, j) < I'(i, j) \end{cases} \end{aligned} \quad (23)$$

W pracach [45][49] przedstawiono inne wersje filtra  $K$ , a mianowicie filtr średkowoprzepustowy (*ang. band-pass filter*).

Teraz chciałbym opisać algorytm FBBMA. Algorytm ten zawiera kilka modyfikacji, które mają poprawiać działanie algorytmu. Opisane zostały w [42]. Te udoskonalenia przedstawiają się następująco:

- **Wykrywanie nieruchomych makrobloków (ang. still macroblock detection)**

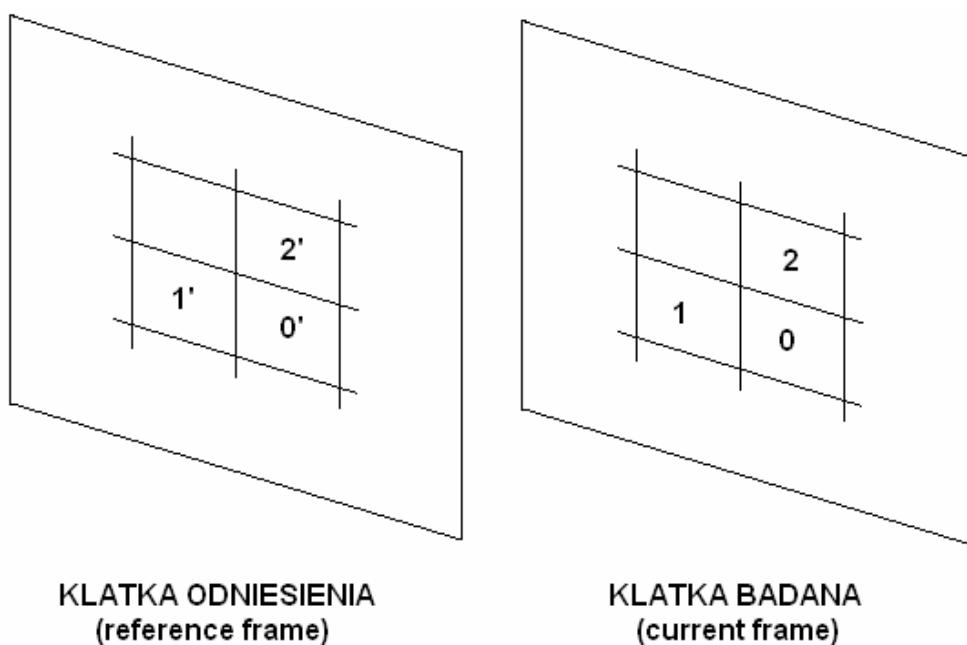
Każdy blok zawiera 16 pod-bloków o rozmiarze  $4 \times 4$  piksele. Jeśli bezwzględna różnica 16 pod-bloków z klatki badanej i odpowiadającym i pod-blokom z klatki odniesienia jest mniejsza od założonego progu  $T_1$  to blok jest nieruchomy. Jeśli bloku nie można określić jako nieruchomy to liczony jest współczynnik SAD. Jeśli SAD bloku z klatki odniesienia i bloku z klatki badanej jest mniejszy od progu  $T_2$  to blok jest ruchomy. W innym przypadku blok jest nieruchomy.

- **Adaptacyjne przewidywanie centralnego punktu szukania (ang. adaptive central-search-point prediction)**

Po raz pierwszy opisano te innowacje w pracach [50]-[53]. Technika ta wykorzystuje czasoprzestrzenną korelację wektorów ruchu do estymowania centralnego punktu szukania. W ogólnym przypadku przewidywanie punktu centralnego sprowadza się do równania:

$$mv0 = \frac{mv0' + [(mv1 - mv1') + (mv2 - mv2')]}{2} \quad (24)$$

gdzie:  $mv0$  jest przewidywanym wektorem ruchu dla obecnie badanego bloku,  $mv1$  i  $mv2$  odpowiednio wektorami ruchu bloków po lewej i górnej stronie badanego bloku. Wektory ruchu  $mv0'$ ,  $mv1'$  oraz  $mv2'$  są wektorami ruchu bloków z klatki odniesienia (patrz rysunek 19).



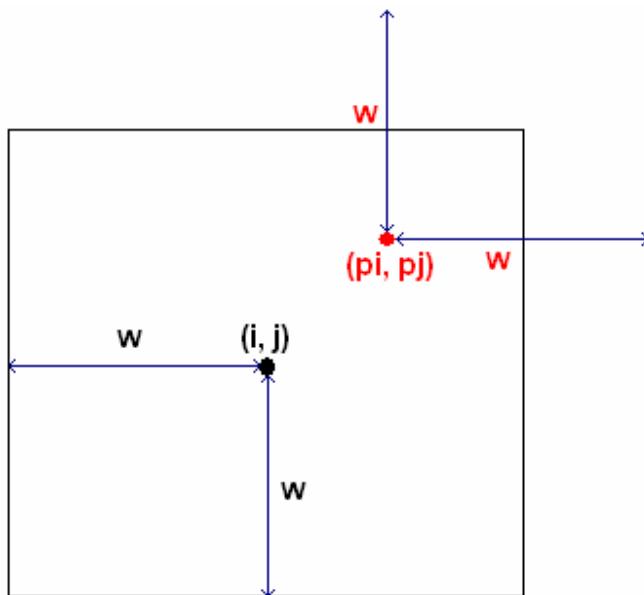
Rysunek 19. Geometria pod-bloków w makrobloku przy przewidywaniu początkowego wektora ruchu

Istnieją dwie modyfikacje tej metody: **centralna selekcja** (ang. search center selection) i **centralne ograniczenie zakresu** (ang. search range limitation).

**Centralna selekcja.** Czasem przewidziany  $mv0$  nie jest dobrym punktem startowym przeszukiwania. Więc centralny punkt ( $px$ ,  $py$ ), w którym zaczniemy przeszukiwanie ustalamy w zależności od współczynnika SAD. A mianowicie liczymy SAD dla  $mv0$  i bloku z pozycji ( $i$ ,  $j$ ), czyli bloku nie przesuniętego. Jeśli wartość współczynnika SAD dla  $mv0$  jest mniejsza niż wartość SAD dla bloku

z pozycji  $(i, j)$  to punktem startowym  $(pi, pj)$  szukania będzie blok z pozycji  $mv0$ . W innym przypadku będzie to blok z pozycji  $(i, j)$ .

**Centralne ograniczenie zakresu.** Założmy, że znaleziony punkt startowy  $(pi, pj)$  znajduje się blisko granic obszaru przeszukiwania. W takiej sytuacji możemy wyeliminować z przeszukiwania bloki znajdujące się poza obszarem przeszukiwania. Sytuację taka ilustruje poniższy rysunek. Obszar przeszukiwania jest oddalony od pozycji  $(i, j)$  o długość  $w$ . Długość  $w$  jest najczęściej wyrażana w szerokościach bloku.



Rysunek 20. Punkt startowy blisko granic obszaru przeszukiwania.

- **Zmienny zasięg przeszukiwania (ang. variable search distance)**

Ogólnie przewidywanie centralnego punktu szukania powoduje znalezienie punktu  $(px, py)$ , który znajduje się w pobliżu optymalnego punktu. Co spowoduje szybsze odszukanie punktu (bloku) optymalnego w procesie przeszukiwania [42]. Niekiedy wyniki takiego przewidywania nie są korzystne. Ma to miejsce gdy wartość zasięgu przeszukiwania jest zbyt mała. Zasięg przeszukiwania  $w$  zależy więc od dopasowania bloku z pozycji  $(pi, pj)$ . Zasięg ten wyraża się wzorem:

$$\begin{cases} \frac{w}{4} \Leftrightarrow SAD(pi, pj) < T3 \\ \frac{w}{2} \Leftrightarrow T3 \leq SAD(pi, pj) < T4 \\ w \Leftrightarrow SAD(pi, pj) \geq T4 \end{cases} \quad (25)$$

- **Szybkie binarne dopasowywanie bloków (ang. fast binary block matching)**

Jak wiadomo współczynnik NNMP porównuje wartości pikseli binarnych a właściwie wykonuje operacje ex-or na pikselach z klatki badanej i odniesienia. W pracy [42] zaproponowano pewną modyfikację współczynnika dopasowania NNMP. Mianowicie w celu zwiększenia szybkości wyliczenia NNMP dla dwóch bloków zmieniono wzór określający ten współczynnik:

$$NNMP = \sum_{i=0}^7 NumOf1s(u(i) \otimes v(i)) \quad (26)$$

gdzie:  $u(i)$  i  $v(i)$  są 32-bitowymi wartościami reprezentującymi wartości binarne bloków z dwóch linii obrazu binarnego.  $NumOf1s$  oznacza liczbę 1 otrzymaną z operacji ex-or na  $u(i)$  i  $v(i)$ . Natomiast  $\otimes$  jest operacją ex-or na 32-bitowych wartościach  $u(i)$  i  $v(i)$ .

- **Szybszy stop (ang. halfway stop)**

Współczynnik NNMP składa się z 8 operacji a właściwe z 8 części. Wyliczając go dla aktualnie badanego bloku możemy zatrzymać te wyliczenia. Jest to metoda szybszego stopu. Licząc NNMP porównujemy kolejne 8 części ze znalezioną wcześniej minimalną wartością NNMP - MNNMP. Jeśli aktualna wartość NNMP jest większa od MNNMP to przerywamy wyliczenia i przechodzimy do obliczeń dla następnego bloku. w innym przypadku wyliczamy NNMP dalej (oczywiście jeśli wyliczenie NNMP nie skończyło się). Tak więc w skrócie można powiedzieć, że metoda ta polega na porównywaniu NNMP w kolejnych jego 8 częściach z MNNMP.

- **Technika wielu kandydatów (ang. multiple candidates selection)**

Mając blok startowy ( $p_i, p_j$ ) oraz bloki znalezione w wyliczeniach NNMP wyliczamy dla nich współczynnik SAD. Ten blok, który ma najmniejszą wartość SAD jest ostatecznym wektorem ruchu.

**Algorytm FBBMA:**

**Krok1.** Klatka dzielona jest na niepokrywające się bloki o rozmiarze 4x4 piksele. Dla każdego bloku liczymy średnią wartość jego pikseli. Następnie przy użyciu transformaty 1BT tworzymy obraz binarny.

**Krok 2.** Wykrywamy nieruchome makrobloki. Jeśli blok jest nieruchomy to wróć wektor ruchu  $\vec{m}(i, j) = (i, j)$  i przejdź do następnego makrobloku. W innym przypadku idź do kroku 3.

**Krok 3.** Za pomocą adaptacyjnego przewidywania centralnego punktu szukania określamy punkt szukania ( $p_i, p_j$ ). Obliczamy  $SAD(p_i, p_j)$  jeśli nie jest to blok z pozycji  $(i, j)$ . Jeśli  $SAD(p_i, p_j)$  jest mniejszy niż  $T_2$  to wróć wektor ruchu  $\vec{m}(i, j) = (p_i, p_j)$  i przejdź do kolejnego bloku. W innym przypadku idź do kroku 4.

**Krok 4.** Obliczamy zasięg przeszukiwania za pomocą metody zmiennego zasięgu przeszukiwania.

**Krok 5.** Stosujemy metody szybkiego binarnego dopasowywania bloków oraz szybszego stopu. Punktem startowym jest  $(p_i, p_j)$ . W efekcie czego otrzymujemy bloki o minimalnym NNMP. Bloki te nazywane są kandydatami.

**Krok 6.** Stosujemy technikę selekcji wielu kandydatów do znalezienie końcowego wektora ruchu.

### 3.5.9. Algorytm PSO-ZMP

Algorytm PSO-ZMP (*ang. Particle Swarm Optimization – Zero-motion Prejudgment*) zaproponowali Ren, Manokar, Shi oraz Zheng w pracy [62]. Wykorzystuje on optymalizację typu particle swarm. Zanim opiszę algorytm PSO-ZMP chciałbym przybliżyć optymalizację particle swarm. Po raz pierwszy ten typ optymalizacji przedstawili Eberhart i Kennedy w pracach [60] [61]. Ciekawostką jest, że algorytm ten powstał jako „efekt uboczny” ich pierwotnych badań. A mianowicie chcieli oni zasymulować graficznie choreografię stada ptaków lub ławicy ryb. Optymalizacja ta powstała w efekcie obserwacji zachowywania się istot żywych, których wspólną cechą jest ich stadne życie. Eberhart i Kennedy zauważyl, że można to wykorzystać do budowy modelu obliczeniowego. Z optymalizacją particle swarm wiążą się następujące oznaczenia i terminy:

- *cząsteczka* (pojedynczy osobnik stada np. ptak)
- *rój* (stado np. stado ptaków)
- $p_i$  – najlepsze położenie pojedynczej cząstki
- $p_g$  – najlepsze położenie podzbioru roju (grupy, np. najlepsze położenie sąsiadów)

Optymalizacja particle swarm składa się z następujących kroków:

- zainicjowanie populacji początkowej (elementami tej populacji są losowe wygenerowane rozwiązania)
- poszukiwanie optimum poprzez modyfikację kolejnych populacji rozwiązań

Cząsteczki poprawiają swoje wartości dopasowania (optymalizują swoją odległość od pożywienia). Wartości dopasowania cząsteczek są oceniane przez funkcję dopasowania oraz prędkości (wyznaczają kierunek ruchu cząsteczek i długość pokonywanego dystansu).

Wprowadźmy dwa równania (prędkości i położenia):

$$(1) \quad v_{id}(t+1) = w \cdot v_{id}(t) + c_1 \cdot r_1 \cdot (p_{id} - x_{id}) \cdot c_2 \cdot r_2 \cdot (p_{gd} - x_{id})$$

$$(2) \quad x_{id}(t+1) = x_{id}(t) + v_{id}(t+1)$$

$$1 \leq i \leq N \quad 1 \leq d \leq D$$

gdzie:

**N** – liczba cząstek

**D** – wymiar cząstki

**V<sub>i</sub>** = (v<sub>i1</sub>, v<sub>i2</sub>, ..., v<sub>ID</sub>) – wektor prędkości cząstki i

**X<sub>i</sub>** = (x<sub>i1</sub>, x<sub>i2</sub>, ..., x<sub>ID</sub>) – wektor pozycji cząstki i

**c<sub>1</sub>, c<sub>2</sub>** – dodatnie stałe (najczęściej c<sub>1</sub> = c<sub>2</sub> = 2.0)

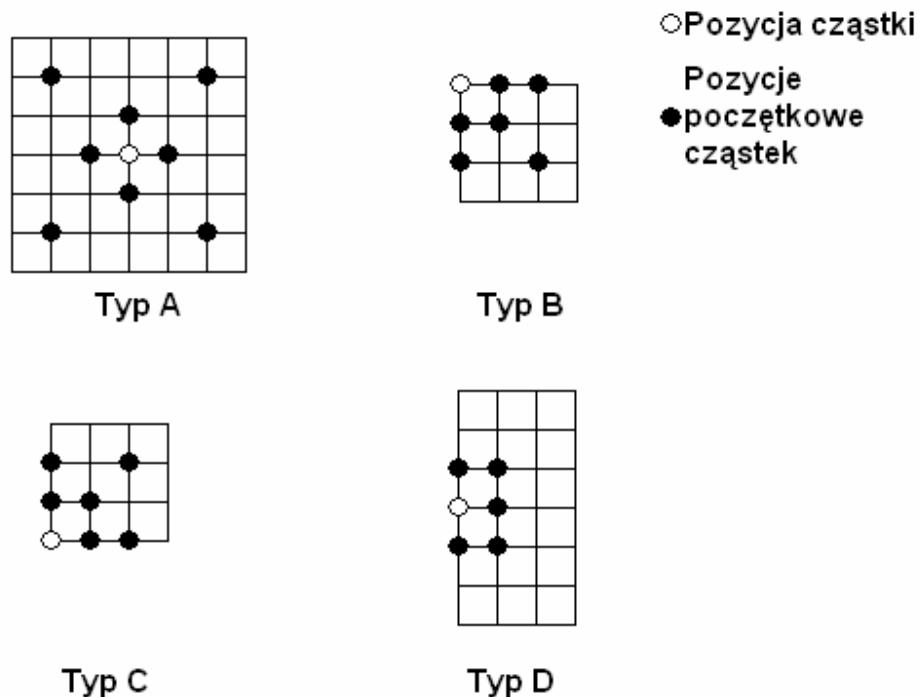
**r<sub>1</sub>, r<sub>2</sub>** – dwie różne zmienne losowe wylosowane z rozkładu jednostajnego

**w** – waga

**p<sub>g</sub>** – najlepsza rozwiązanie znalezione do tej pory dla grupy

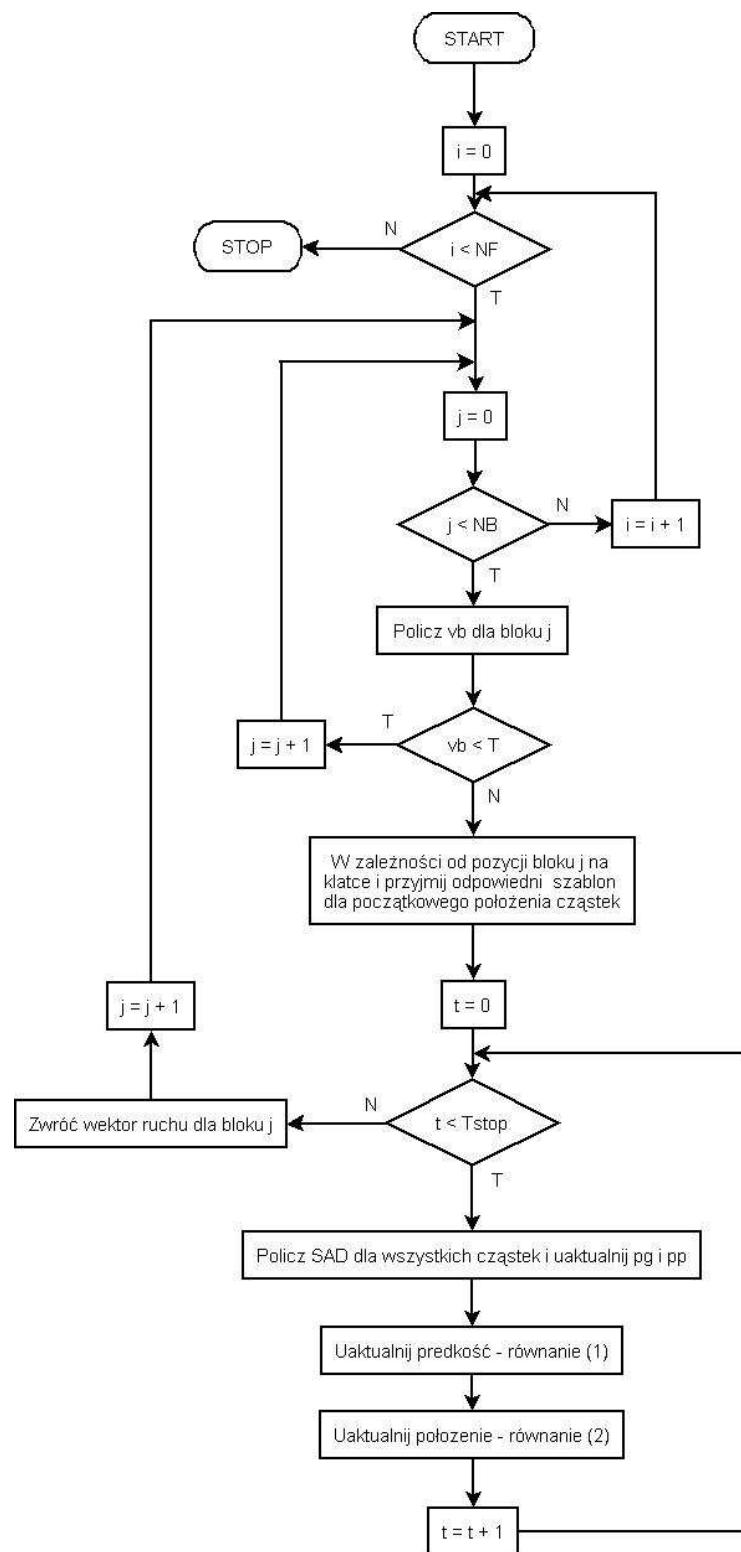
**p<sub>i</sub>** – najlepsze rozwiązanie znalezione do tej pory dla cząstki

W algorytmie PSO-ZMP położenie początkowej cząstek (w naszym przypadku jest to blok) zależy od położenia aktualnie badanego bloku na klatce (rysunek). Istnieją cztery różne szablony według, których cząstka przyjmuje swoją pozycję początkową.



**Rysunek 21.** Szablony według których cząstki przyjmują swoje położenie początkowe.

Termin ZMP (ang. *zero motion prejgment*) oznacza metodę, która powoduje wyeliminowanie nieruchomych bloków z procesu przeszukiwania. Polega to na porównywaniu obliczonego współczynnika dopasowania z progiem  $T$ . Jeśli wartość tego współczynnika jest mniejsza/równa od  $T$  to odrzucamy taki blok (wektor ruchu wynosi  $(i, j)$ ).



Rysunek 22. Algorytm PSO-ZMP

Na rysunku widać działanie algorytmu PSO-ZMP. Oznaczenia przyjęte w opisie algorytmu:

**i** – i-klatka filmu (i-obraz sekwencji obrazów)

**j** – j-blok

**NF** – liczba klatek w filmie

**NB** – liczba bloków

**vb** – współczynnik dopasowania (np. SAD) dwóch bloków (bloków z pozycji j na klatce i oraz klatce odniesienia i - 1)

**t** – iteracja optymalizacji particle swarm

**Tstop** – warunek stopu dla optymalizacji particle swarm

**pp** – najlepsze dopasowanie dla cząstki

**pg** – najlepsze dopasowanie cząstki w grupie

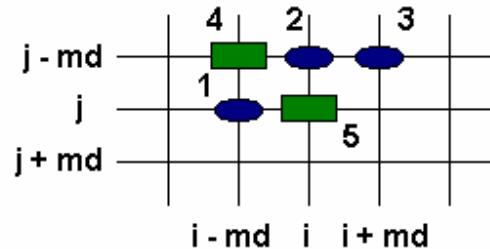
Warunkiem stopu *Tstop* w tej wersji algorytmu jest ilość iteracji. Natomiast można przyjąć inny warunek stopu (np. wartość dopasowania cząstki).

### 3.5.10. Algorytm PSA

Algorytm PSA (ang. *Prediction Search Algorithm*) został opisany przez Luo, Zou i Gao w pracy [64]. Algorytm ten wykorzystuje liniowe korygowanie wektorów ruchu trzech sąsiednich bloków do uzyskania przewidywanego wektora ruchu nazywanego początkowym punktem przeszukiwania (ang. *initial search point*). Za pomocą ruchomego okna przeszukującego o rozmiarze 3x3 bloków i środka w punkcie początkowym kontynuowane jest przeszukiwanie. Proces przeszukiwania prowadzony jest ze stałym krokiem szukania S i trwa dopóki ruchome okno osiągnie granicę okna przeszukiwania lub blok o najlepszym dopasowaniu znajduje się w centrum ruchomego okna przeszukującego.

Początkowy punkt przeszukiwania przewidywany jest na pozycjach sąsiednich bloków (rysunek). Używamy tylko wektorów ruchu bloków 1,2 i 3. Blok 5 jest blokiem

aktualnie badanym. Pozycja 4 jest wykluczana z powodu dużej zależności od bloków 1 i 2 [64].



**Rysunek 23.** Rozkład bloków używanych do przewidywania początkowego punktu przeszukiwania ( $md$  – maksymalne przesunięcie)

Założymy, że

$$V_k = [V_{1k}, V_{2k}]^T \quad k = 1, 2, 3 \quad (27)$$

reprezentuje wektory ruchu trzech bloków (zgodnie z rysunkiem). Natomiast

$$P = [P_1, P_2]^T \quad (28)$$

oznacza przewidywany wektor ruchu aktualnie badanego bloku. A

$$W = [W_1, W_2, W_3]^T \quad (29)$$

jest wektorem wagowym. Wtedy początkowy punkt przeszukiwania (początkowy wektor ruchu dla badanego bloku):

$$W = [W_1, W_2, W_3]^T = \left[ \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right]^T$$

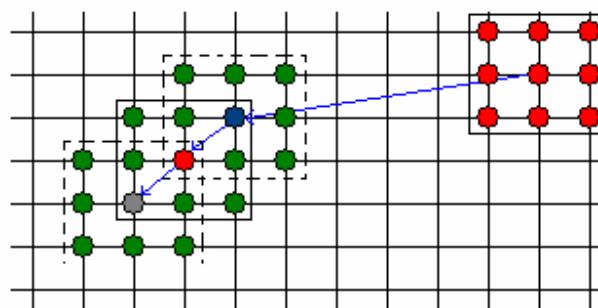
$$P = [P_1, P_2]^T = \left[ \sum_{k=1}^3 W_k V_{1k}, \sum_{k=1}^3 W_k V_{2k} \right]^T \quad (30)$$

**Algorytm PSA:**

**Krok 1:** Za pomocą równania (30) uzyskaj pozycję początkowego punktu przeszukiwania, który jest początkowym centralnym punktem ruchomego okna przeszukującego. Policz współczynniki dopasowania dla bloków z okna przeszukującego. Jeśli minimalną wartość współczynnika dopasowania posiada blok z środkowej pozycji okna to idź do kroku 3. W innym przypadku krok 2.

**Krok 2:** Przesuń okno na pozycję ostatnio znalezionej bloku o najlepszym dopasowaniu. Policz współczynniki dopasowania dla 9 bloków wchodzących w skład okna przeszukującego. Jeśli najlepsze dopasowanie posiada blok z pozycji środkowej to idź do kroku 3. W innym przypadku idź do kroku 2.

**Krok 3:** Zwróć wektor ruchu dla badanego bloku. Wektorem ruchu dla aktualnie badanego bloku jest pozycja ostatnio znalezionej bloku o najlepszym dopasowaniu.



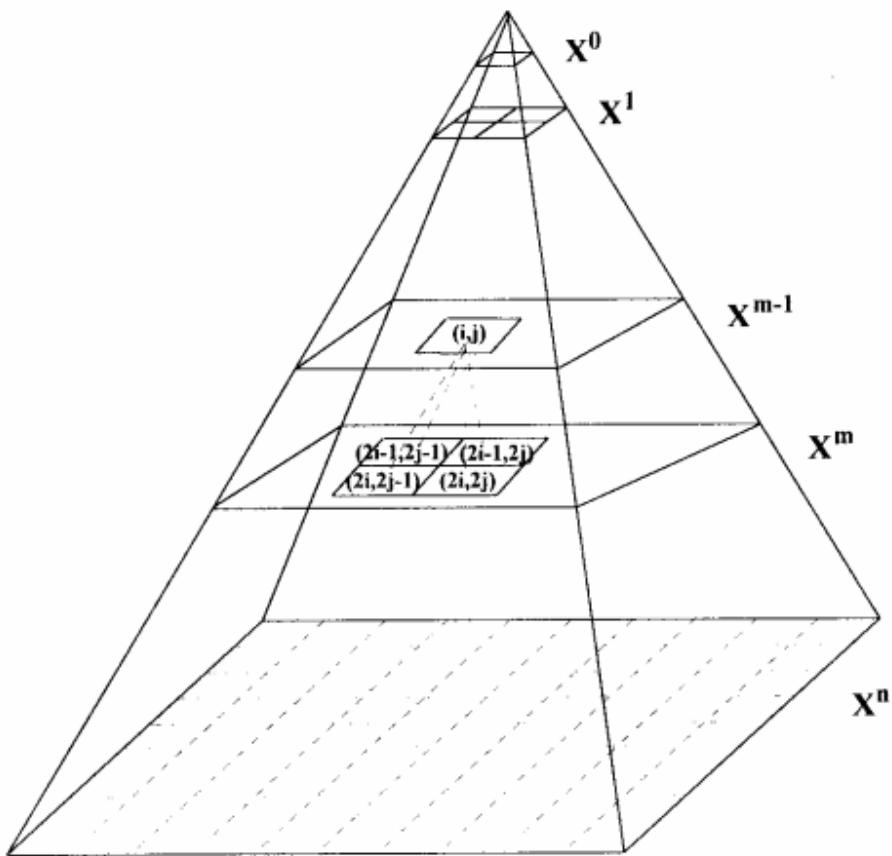
Rysunek 24. Przykład działania algorytmu PSA

Na rysunku widać działanie algorytmu. Najpierw przewidziano położenie początkowego punktu przeszukiwania (niebieski punkt). Następnie policzono współczynniki dopasowania dla bloków z ruchomego okna przeszukiwania. Najlepsze dopasowanie posiadał blok w lewym dolnym rogu okna. Następnie

ruchome okno przeszukiwania przesunęło się na pozycje ostatnio znalezionego bloku o najlepszym dopasowaniu. Znowu policzono wartości współczynników dopasowania dla bloków z okna. Ponownie najlepszy okazał się blok w lewym dolnym rogu okna. Okno przesunęło się na pozycje bloku o najlepszym dopasowaniu. Następnie policzono współczynniki dopasowania dla bloków wchodzących w skład okna. Blok o najlepszym dopasowaniu znajdował się w centralnej pozycji okna (szary punkt) więc algorytm zakończył działanie. Wektorem ruchu dla aktualnie badanego bloku jest pozycja bloku znajdującego się na pozycji oznaczonej szarym punktem.

### 3.5.11. Algorytm BSPA

Algorytm BSPA (*ang. Block Sum Pyramid Algorithm*) opisany został przez Lee i Chen w pracy [66]. Algorytm ten eliminuje niepotrzebne pozycje przeszukujące. Na początku budujemy sumacyjną strukturę piramidalną bloku (*ang. sum pyramid structure of a block*). Pozycje są eliminowane wraz z zagłębianiem się w niższe poziomy piramidy.



Rysunek 25. Piramidalna struktura danych [66].

Założymy, że każdy blok klatki jest rozmiaru  $N \times N$  gdzie  $N = 2^n$  i  $n$  jest dodatnią liczbą całkowitą. Dla każdego bloku  $X$  odpowiadająca mu piramida jest sekwencją bloków  $\{X^0, \dots, X^{m-1}, X^m, X^{m+1}, \dots, X^n\}$ . Istnieje pewna zależność:

$$X^{m-1}(i, j) = f[X^m(2i - 1, 2j - 1), X^m(2i - 1, 2j), X^m(2i, 2j - 1), X^m(2i, 2j)] \quad (31)$$

gdzie  $f$  jest funkcją sumacyjną (wartość poziomu szarości każdego piksela jest sumą wartości sąsiedztwa  $2 \times 2$  pikseli na następnym poziomie) a więc:

$$X^{m-1}(i, j) = X^m(2i - 1, 2j - 1) + X^m(2i - 1, 2j) + X^m(2i, 2j - 1) + X^m(2i, 2j) \quad (32)$$

Wprowadźmy współczynnik dopasowania  $MAD^m$  jako:

$$MAD^m(X, Y) = MAD(X^m, Y^m) = \sum_{j=1}^{2^m} \sum_{h=1}^{2^m} |X^m(j, h) - Y^m(j, h)| \quad (33)$$

gdzie  $X^m(j, h)$  i  $Y^m(j, h)$  oznaczają wartość piksela z pozycji  $(j, h)$  odpowiednio w bloku  $X^m$  i  $Y^m$ . Wprowadźmy pojęcie sumy wartości pikseli bloku jako:

$$S_x = \sum_{i=1}^{2^m} \sum_{j=1}^{2^m} |X(i, j)| \quad (34)$$

### Algorytm BSPA

Liczymy sumę piramidalną bloków z klatki odniesienia, które odpowiadają aktualnie przeszukiwanej pozycji na klatce badanej. Ustalamy sumę piramidalną bloku tymczasowego  $T$  (blok na aktualnym poziomie piramidy). Następnie liczymy MAD pomiędzy blokiem  $T$  i blokiem o wartości wektora ruchu  $(i, j)$ . Otrzymaną wartość nazwijmy  $MAD_{min}$ . Dla pozostałych bloków z  $X$  liczymy MAD na wyższym poziomie  $MAD^0(T, X)$ . Jeśli  $MAD^0(T, X)$  jest mniejsza od  $MAD_{min}$  to uznajemy ten blok za blok o najlepszym dopasowaniu i zwracamy jego pozycję jako wektor ruchu. W innym przypadku robimy to samo dla poziomu 1. Powtarzamy tą czynność dopóki nie znajdziemy bloku o najlepszym dopasowaniu.

#### 3.5.12. Algorytm FEBMEA

Yue, Jian, Yiliang, Fengting i Chenghui zaproponowali w pracy [67] algorytm FEBMEA (ang. fast effective block motion estimation algorithm). Proponują oni użycie współczynnika dopasowania MAD, a także podział pikseli wchodzących w skład danego bloku na cztery grupy. Grupy pikseli to  $a, b, c, d$ , a sposób podziału bloku widać na rysunku.

a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d

Rysunek 26. Podział pikseli na cztery grupy.

4		1		4		1		
3		2		3		2		
				S				
4		1		4		1		
3		2		3		2		

Rysunek 27. Podział okna przeszukiwania

**Algorytm FEBMEA:**

- Krok 1:** Podziel wszystkie piksele bloku na cztery grupy (według rysunku).
- Krok 2:** Policz współczynniki MAD dla bloków z okna przeszukiwania w zależności od grupy (rysunek).
- Krok 3:** Policz współczynniki MAD dla najlepszych bloków z danej grupy. W tym kroku współczynniki MAD liczone są dla wszystkich pikseli z bloku.

**Krok 4:** Policz współczynniki MAD dla ośmiu sąsiednich bloków dla bloku, który miał najlepsze dopasowanie w kroku 3. Blok o najmniejszym współczynniku MAD jest wektorem.

Autorzy sugerują, że krok 4 jest zbędny gdyż uzyskane przez nich wyniki były zadowalające także bez tego kroku [67].

Należy zwrócić uwagę na to, że w zależności od pozycji bloku w oknie przeszukiwania (1, 2, 3 lub 4) liczymy MAD tylko dla niektórych pikseli z bloku. Tak więc jeśli blok znajduje się na pozycji 1 to liczymy MAD tylko dla pikseli z grupy a. Analogicznie jest dla pozostałych pozycji, a więc dla pozycji 2 bierzemy pod uwagę piksele z grupy b itd.

### 3.5.13. Algorytm VSS

Algorytm VSS (*ang. variable-step search algorithm*) opisali Kwatra, Liu, Whyte w pracy [69]. Innymi pracami, w których można przeczytać o VSS to [68] [70]. Zdefiniujmy

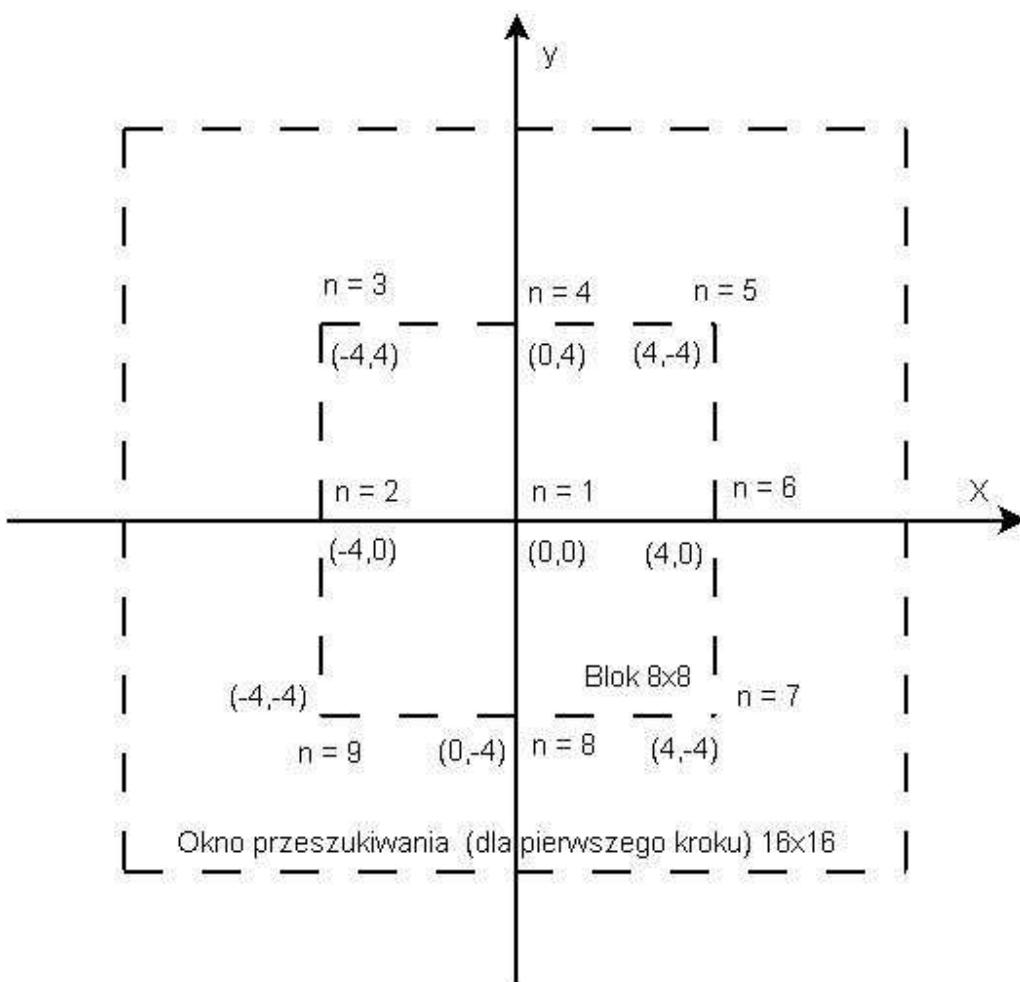
$$d(m,n) = \text{MAD}(x, y) \quad (35)$$

gdzie:

**m** – numer kroku,  $m = 1, 2, \dots, r$

**n** – pozycja w oknie przeszukiwania (takie jak w TSS),  $n = 1, 2, \dots, 9$ . Pozycje w oknie przeszukiwania przedstawia rysunek.

**r** – ostatni krok



**Rysunek 28.** Okno przeszukiwania w pierwszym kroku

### Algorytm VSS:

- Krok 1:** Jeśli  $d(1,1) \leq T_1$  to zakończ przeszukiwanie (blok nieruchomy).
- Krok 2:** Jeśli  $d(1,1) > T_1$  to policz  $d(1,2)$ ,  $d(1,3)$ , ...,  $d(1,9)$ .
- Krok 3:** Jeśli  $\min_{1 \leq n \leq 9} d(1,n) \leq T_1$  to zakończ szukanie i zwróć tę pozycję bloku jako wektor ruchu.
- Krok 4:** Jeśli  $\min_{1 \leq n \leq 9} d(1,n) > T_1$  to ustaw nowy punkt centralny w pozycji bloku  $\min_{1 \leq n \leq 9} d(1,n)$ . Ustal  $\min_{1 \leq n \leq 9} d(2,n)$ .
- Krok 5:** Jeśli  $\min_{1 \leq n \leq 9} d(2,n) \leq T_1$  to zakończ szukanie i zwróć  $\min_{1 \leq n \leq 9} d(1,n)$  jako wektor ruchu. W innym przypadku krok 6.

**Krok 6:** Jeśli  $\min_{1 \leq n \leq 9} d(2, n) > T_1$  i  $\left| \min_{1 \leq n \leq 9} d(1, n) - \min_{1 \leq n \leq 9} d(2, n) \right| \leq T_2$  to oznacza zbyt duży wpływ szumów na przeszukiwanie [68], a więc zwróć  $\min_{1 \leq n \leq 9} d(2, n)$  jako wektor ruchu. W innym przypadku krok 7.

**Krok 7:** Jeśli  $\min_{1 \leq n \leq 9} d(2, n) > T_1$  i  $\left| \min_{1 \leq n \leq 9} d(1, n) - \min_{1 \leq n \leq 9} d(2, n) \right| > T_2$  to krok 8.

**Krok 8:** Jeśli kolejny krok będzie większy od ustalonego ostatniego kroku r to zwróć jako wektor ruchu  $\min_{1 \leq n \leq 9} d(r, n)$

Oczywiście w kolejnych powrotach do kroku 3 odpowiednio zmieniają się warunki np.

$$\left| \min_{1 \leq n \leq 9} d(2, n) - \min_{1 \leq n \leq 9} d(3, n) \right| \leq T_2$$

Progi  $T_1$  oraz  $T_2$  musimy ustalić sami.

### 3.5.14. Algorytm CDS

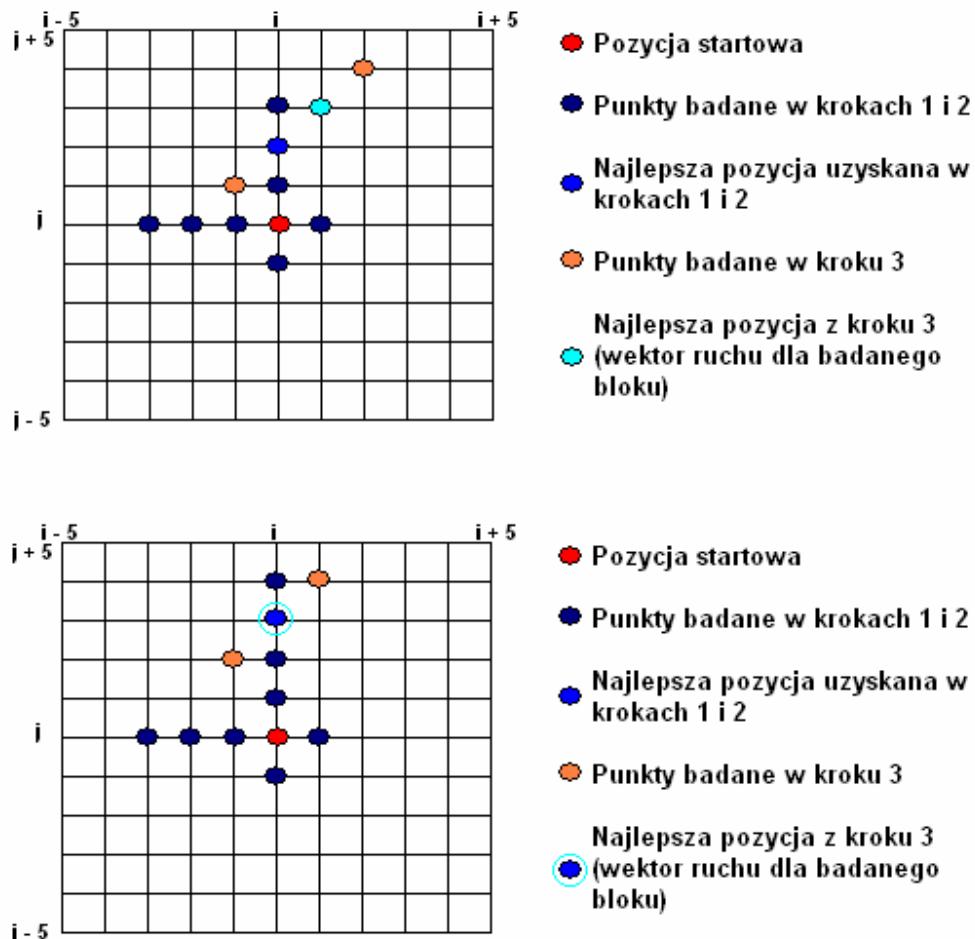
Metodę CDS (*ang. conjugate direction search*) po raz pierwszy opisali Rao i Srinivasan w [71]. Natomiast Lin i Sun w [68] zaproponowali algorytm dopasowania bloków oparty na metodzie CDS. W [68] zaproponowano i zalecono użycie MAD jako współczynnika dopasowania dla bloków. Algorytm przedstawia się następująco:

**Krok 1:** Policz współczynniki dopasowania  $v_b$  dla bloków  $B(i, j)$ ,  $B(i+1, j)$ ,  $B(i-1, j)$ . Jeśli najlepsze dopasowanie posiada blok z pozycji  $B(i+1, j)$  to liczymy dopasowanie dla bloku  $B(i+2, j)$ . Poruszamy się w kierunku poziomym w stronę bloku o minimalnej wartości. Liczymy dopasowania dla bloków  $B(i, j)$ ,  $B(i+1, j)$ ,  $B(i+2, j)$ . Kolejne wyliczenia dopasowania bloków znajdujących się na drodze w kierunku poziomym kończymy gdy dwa sąsiednie bloki bloku o minimalnym dopasowaniu mają większe wartości dopasowania. W ten sposób otrzymujemy najlepsze położenie  $i$ . UWAGA. Gdy blok z pozycji  $B(i, j)$  ma najlepsze dopasowanie to od razu przechodzimy do kroku 2.

**Krok 2:** Analogicznie do kroku 1 poruszamy się w kierunku pionowym. Uzyskujemy w ten sposób najlepsze położenie  $j$ .

**Krok 3:** Przeszukiwanie zaczynamy od „najlepszej” pozycji uzyskanej w krokach 1 i 2. Jeśli tą pozycją startową nie zmieniła tzn. jest znajduje się ona w  $(i, j)$  to kończymy przeszukiwanie – blok jest nieruchomy. Jeśli najlepszą pozycją jest  $(i+2, j+2)$  to teraz szukamy minimum w ukośnym kierunku analogicznie jak w krokach 1 i 2. A więc liczymy dopasowania bloków z pozycji  $(i+1, j+1)$ ,  $(i+2, j+2)$  i  $(i+3, j+3)$ . Przeszukiwanie kończymy gdy minimalna wartość dopasowania znajduje się pomiędzy dwoma większymi wartościami współczynników dopasowania. Innymi słowy jeśli sąsiednie bloki danego bloku mają gorsze dopasowania.

Przykład działania algorytmu CDS pokazano na rysunku. Znajdują się tam dwa przypadki, które pokazują dwie odmienne sytuacje. Różnica polega na tym, że najlepsze pozycje uzyskane w krokach 1 i 2 reprezentują dwa odmienne przypadki. W pierwszej sytuacji pozycja ta znajduje się w pozycji oddalonej o tą samą odległość w kierunku  $i$  jak i  $j$ . W drugiej sytuacji ta odległość jest różna.



Rysunek 29. Przykład działania algorytmu CDS

### 3.5.15. Algorytm AMT

Zaproponowany przez Xu, Man i Cheung w pracy [73] algorytm AMT (*ang. adaptive motion tracking*) jest tak naprawdę udoskonaleniem dla innych algorytmów takich jak TSS, 4SS czy BBGDS. Algorytm ten przyspiesza działanie szybkich algorytmów przeszukiwania i dopasowywania bloków poprzez ustalanie (przewidywanie) punktu startowego.

$B_2(i - 1, j - 1)$	$B_3(i - 1, j)$	$B_4(i - 1, j + 1)$
$B_1(i, j - 1)$	$B_0(i, j)$	

Rysunek 30. Sąsiedztwo badanego bloku  $B_0$ .

Zdefiniujmy średni wektor ruchu dla czterech bloków sąsiednich (rysunek):

$$v_m = \frac{1}{4} \sum_{i=1}^4 v_{Bi} \quad (36)$$

gdzie  $v_{Bi}$  jest współczynnikiem dopasowania danego bloku  $B_i$ .

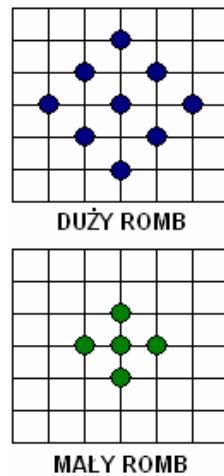
Przyjmujemy pewien próg  $T$ . Punktem startowym może być blok badany oraz jeden z bloków z jego sąsiedztwa. Punkt startowy jest wyznaczany z:

$$V_{init} = \begin{cases} B_i \Leftrightarrow |v_{Bi} - v_m| < T \\ B_0 \Leftrightarrow |v_{B_0} - v_m| \geq T \end{cases} \quad (37)$$

gdzie  $V_{init}$  jest punktem startowym.

### 3.5.16. Algorytm DS

Algorytm DS (*ang. diamond search*) opisali Zhu i Ma w [13]. Algorytm używa dwóch szablonów przeszukiwania w kształcie rombu. Szablony nazywane są DR (Duży Romb) i MR (Mały Romb).



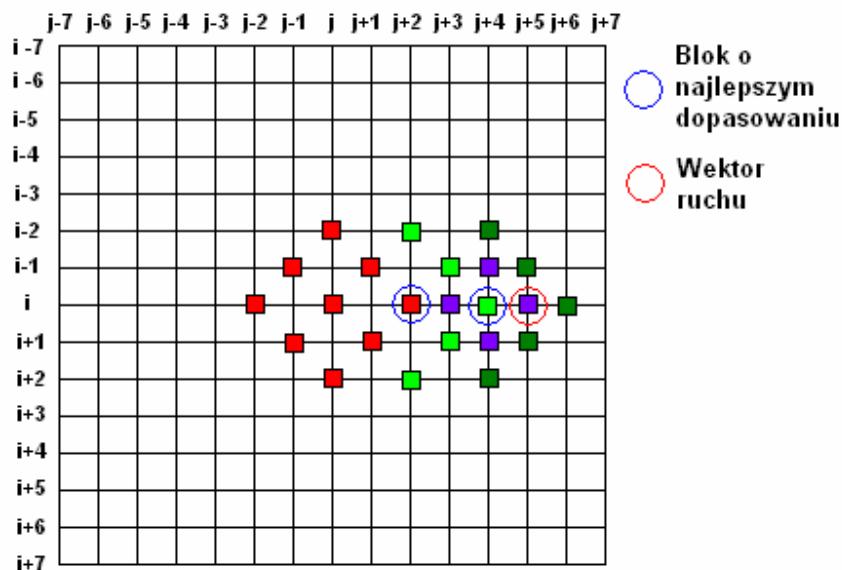
Rysunek 31. Duży i mały romb wykorzystywany w algorytmie DS

### Algorytm DS

**Krok 1:** Policz współczynniki dopasowania dla bloków tworzących DR z pozycją centralną w  $(i, j)$ . Jeśli najlepiej dopasowanym blokiem jest blok z pozycji centralnej DR to zakończ przeszukiwanie – blok nieruchomy (wektor ruchu  $\vec{m}(i, j) = (i, j)$ ). W przeciwnym wypadku krok 2.

**Krok 2:** Ustaw pozycję centralną  $P$  na pozycji najlepiej dopasowanego bloku z poprzedniego kroku. Policz współczynniki dopasowania dla bloków tworzących DR z pozycją centralną w  $P$ . Jeśli najlepiej dopasowanym blokiem jest blok z pozycji centralnej  $P$  to idź do kroku 3. W przeciwnym wypadku idź do kroku 2.

**Krok 3:** Ustaw pozycję centralną  $P$  na pozycji najlepiej dopasowanego bloku z poprzedniego kroku. Policz współczynniki dopasowania dla bloków tworzących MR z pozycją centralną w  $P$ . Pozycja bloku o najlepszym dopasowaniu jest szukanym wektorem ruchu dla bloku z pozycji  $(i, j)$ .



Rysunek 32. Przykład działania algorytmu DS

Na rysunku 32 widać działać działanie algorytmu DS. Na początku policzono współczynniki dopasowania dla bloków z DR (bloki czerwone) z pozycją centralną w

$(i, j)$ . Najlepiej dopasowanym blokiem był blok z pozycji  $(i, j+2)$ . Następnie liczymy współczynniki dopasowania dla bloków z DR (bloki jasno-zielone) z pozycją centralną w  $(i, j+2)$ . Najlepiej dopasowanym blokiem był blok z pozycji  $(i, j+4)$ . Następnie liczymy współczynniki dopasowania dla bloków z DR (bloki ciemno-zielone) z pozycją centralną w  $(i, j+4)$ . Najlepiej dopasowanym blokiem jest blok z pozycji  $(i, j+4)$ , czyli blok z pozycji centralnej. Więc teraz liczymy współczynniki dopasowania dla bloków z MR (bloki fioletowe) z pozycją centralną w  $(i, j+4)$ . Najlepiej dopasowanym blokiem jest blok z pozycji  $(i, j+5)$  – wektor ruchu dla bloku z pozycji  $(i, j)$  wynosi  $\vec{m}(i, j) = (i, j + 5)$ .

### **3.6. Zastosowania metod dopasowywania bloków do wykrywania ruchu – przegląd literatury**

W [54] Brad i Letia opisują zastosowania algorytmu dopasowywania bloków do wykrywania ruchu chmur na zdjęciach satelitarnych. Przy użyciu zmodyfikowanego algorytmu blokowego śledzą oni ruch chmur na zdjęciach satelitarnych. Taki system wykrywania ruchu jest pomocny przy prognozowaniu pogody. Pozwala określić, w którym kierunku przemieszczają się chmury. Satelity geostacjonarne dostarczają zdjęcia zwykle w odstępach półgodzinnych.

Di Stefano i Viarani w [55] zaproponowali zastosowanie BMA (*ang. block matching algorithm*) do śledzenia toru ruchu pojazdów przy monitorowaniu ruchu ulicznego. Swoją metodę kierują głównie do zastosowań nocnych, czyli wtedy gdy jest utrudnione obserwowanie obiektów.

W [76] Bagani i Zoratti opisali zastosowanie metod wykrywania ruchu za pomocą dopasowywania bloków w samochodach firmy Spartan. Inżynierowie tej firmy zastosowali układy obserwowania ruchu dookoła samochodu. Ma ta kilka celów m. in. parkowanie, cofanie i przede wszystkim wykrywanie nagle pojawiających się przeszkód na drodze samochodu. Przeszkodami na drodze samochodu mogą być przedmioty martwe (np. inne auta, duże kamienie itp.) jak i żywe (np. ludzie czy

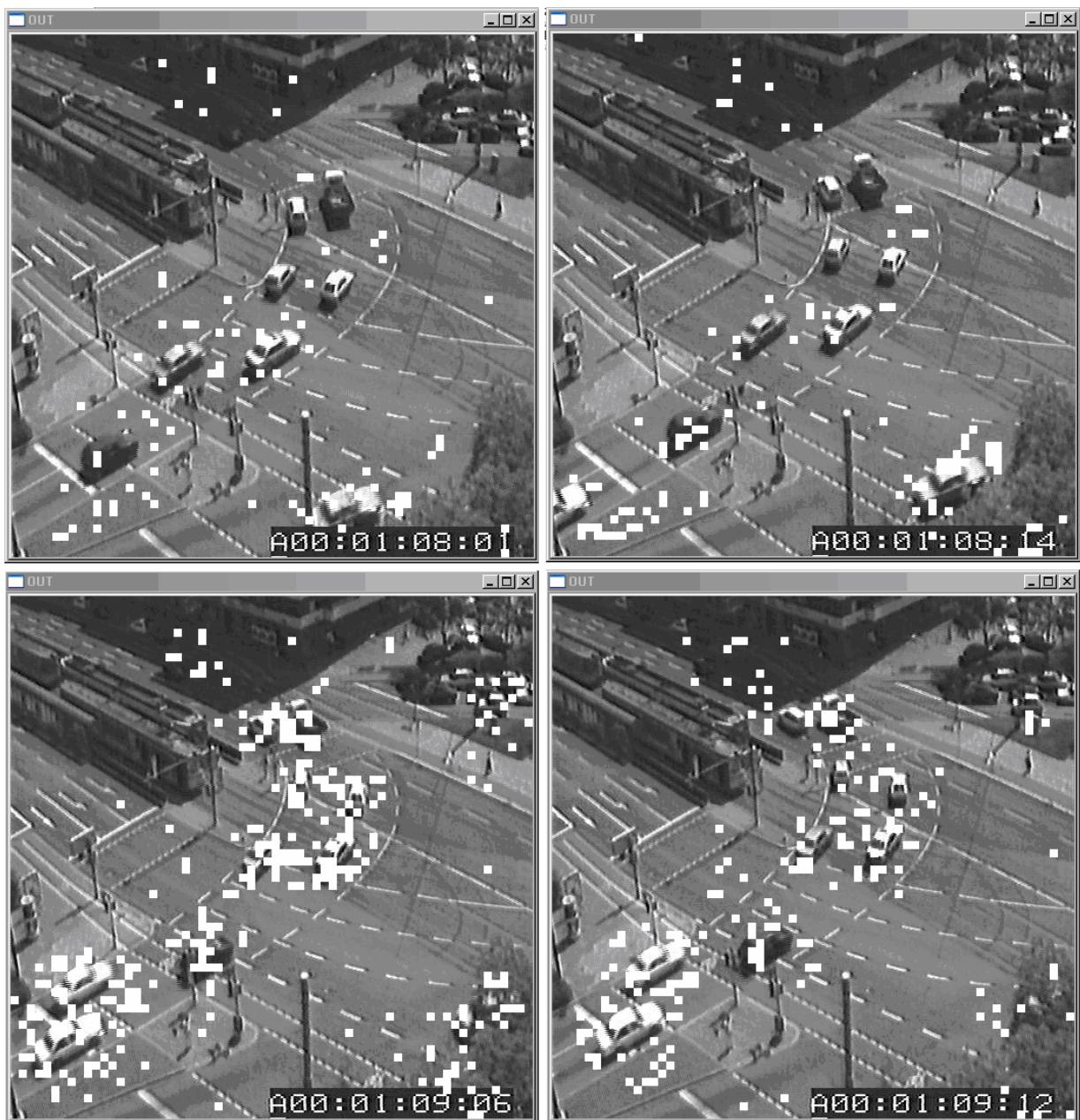
zwierzęta). Takie układy poprawiają bezpieczeństwo nie tylko kierowcy i osób znajdujących się w aucie ale także innych uczestników ruchu. System taki nie pozwoli na wiele niebezpiecznych manewrów jakie mógłby wykonać kierowca np. zbyt mała odległość przy wyprzedzaniu rowerzysty lub zatrzymanie auta przy nagłym wtargnięciu pieszego na jezdnię. W skład systemu wizjnego wchodzi kamera VGA (przesyła 752x480 pikseli z częstotliwością 30 Hz, czyli 30 klatek/sek). Klatki dzielone są na bloki 4x4 pikseli a jako współczynnik zastosowano SAD. Sercem układu wykrywania ruchu jest procesor 32-bitowy DSP.

## **4. Implementacje wybranych algorytmów**

Aplikacje, w których zaimplementowałem niektóre z opisanych algorytmów zostały napisane w środowisku Visual C++ 2005 Express Edition. Projekty aplikacji znajdują się na dołączonej płycie CD. W pliku README.txt znajdującym się na tej płycie jest opisany sposób uruchamiania aplikacji. Wnioski z przeprowadzonych testów znajdują się w rozdziale 5.

### **4.1. Implementacja algorytmu 4SS**

Działanie algorytmu 4SS widać na rysunku 33. Jak można zauważyc algorytm ten nie działa idealnie. Jest narażony na szумy. Aby w pewnym stopniu uczynić ten algorytm odpornym na szumy wprowadziłem pewien współczynnik  $c$ . Ten współczynnik ma za zadanie faworyzować blok centralny w danym kroku algorytmu. Dzieje się tak ponieważ wartość współczynnika dopasowania dla danego bloku jest mnożona przez współczynnik  $c$ . Jako współczynnik dopasowania zastosowałem współczynnik MAD. W podanym przykładzie współczynnik  $c$  wynosi 0.8.



Rysunek 33. Przykład działania algorytmu 4SS. Algorytm zaimplementowany przez autora.

#### 4.2. Implementacja algorytmu TSS

Na rysunku 34 widać działanie zaimplementowanego algorytmu TSS.



Rysunek 34. Przykład działania algorytmu TSS. Algorytm zaimplementowany przez autora.

### 4.3. Implementacja algorytmu CS

Na rysunku 35 widać działanie algorytmu zaimplementowanego przez autora. Zastosowałem MSD jako współczynnik dopasowania. Kolorem żółtym zaznaczone są ruchome bloki.



Rysunek 35. Przykład działania algorytmu CS zaimplementowanego przez autora.

### 4.4. Implementacja algorytmu LOGS

Na rysunku 36 przedstawiono zaimplementowany algorytm LOGS z MSD jako współczynnikiem dopasowania.



Rysunek 36. Przykład działania algorytmu LOGS zaimplementowanego przez autora.

#### 4.5. Implementacja algorytmu BBGDS

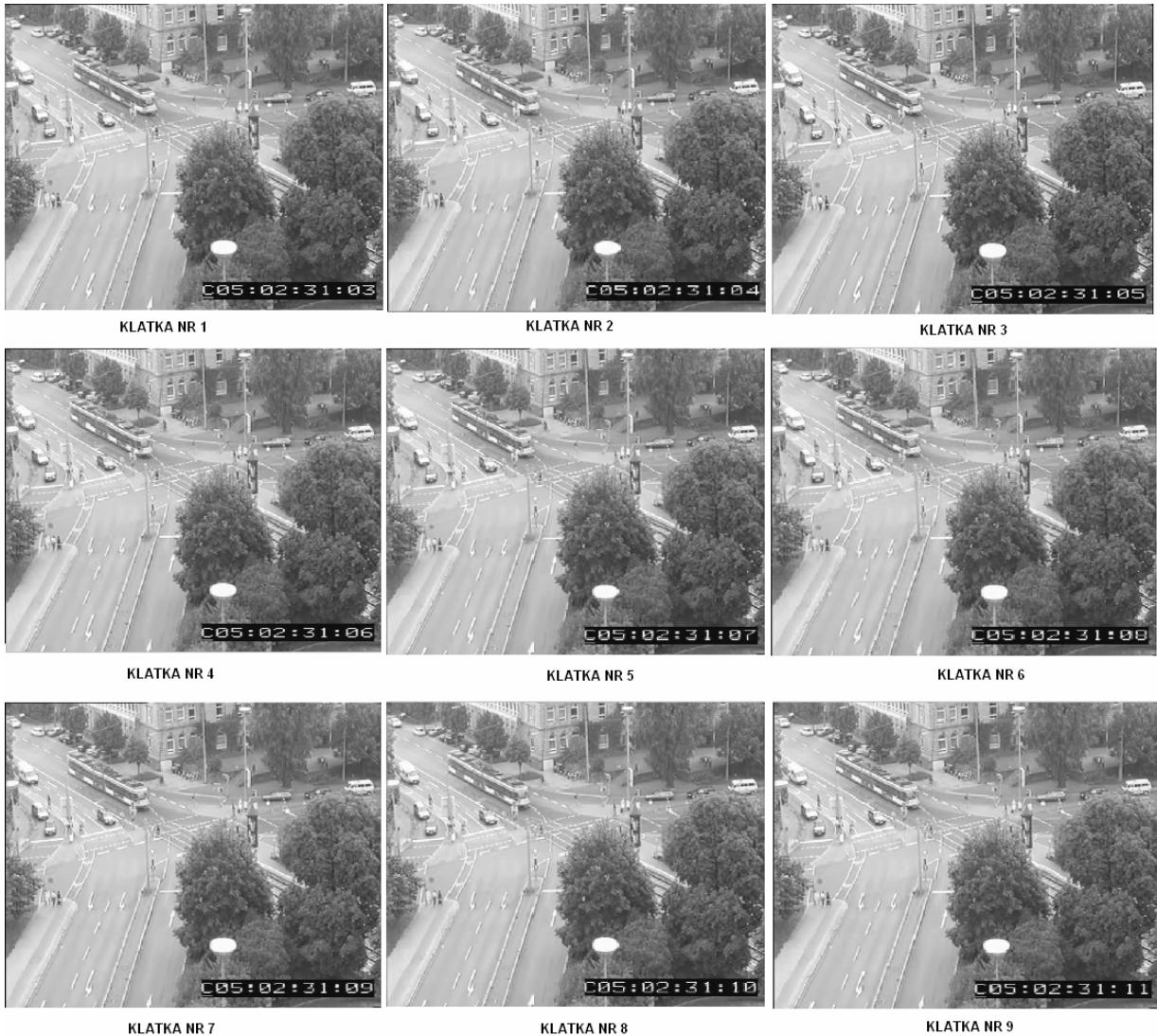
Poniżej prezentuje (na rysunku 37) działanie algorytmu BBGDS. Jego implementacja jest przykładem błędного działania algorytmu wykrywania ruchu. Osiąga on najgorsze rezultaty. Jako współczynnik dopasowania zastosowałem MSD.



Rysunek 37. Przykład działania zaimplementowanego przez autora algorytmu BBGDS.

## 5. Wyniki testów

Do testów użyłem filmu w formacie avi. Wyniki działania algorytmów rejestrowałem dla dziewięciu kolejnych klatek (rysunek 38). Testowałem zaimplementowane algorytmy, które opisałem w rozdziale 4.



**Rysunek 38.** Kolejne 9 klatek filmu używanego do testowania zaimplementowanych algorytmów.

Jako pierwszy przeprowadziłem test, w którym badałem liczbę ruchomych bloków wykrywanych przez dany algorytm. Wyniki uzyskane podczas testów przedstawiłem w tabeli 1.

	<b>Liczba ruchomych bloków w poszczególnych klatkach filmu</b>								
	1	2	3	4	5	6	7	8	9
<b>TSS (SAD)</b>	25	31	46	33	31	23	64	33	17
<b>BBGDS(MSD)</b>	46	35	48	35	48	27	57	41	40
<b>CS (MSD)</b>	109	113	139	114	118	135	163	110	152
<b>LOGS (MSD)</b>	180	173	250	153	181	182	264	160	290
<b>4SS (MAD)</b>	27	23	35	42	27	27	41	37	37

Tabela 1. Liczba wykrytych ruchomych bloków przez poszczególne algorytmy.

Jak widać z powyższej tabeli najmniej ruchomych bloków wykrył algorytm TSS. Biorąc pod uwagę ocenę wizualną działania algorytmu stwierdzam, że TSS wykrył jedynie poruszający się tramwaj. Należy także zwrócić uwagę na to, że dość znaczna część bloków została błędnie rozpoznana. Innymi słowy – bloki wskazane przez algorytm jako ruchome nie zawsze nimi były. Zastosowany w tym algorytmie współczynnik SAD także miał dość istotny wpływ na dużą wrażliwość tego algorytmu na szумy. Po przeprowadzeniu tego testu stwierdzam, że algorytm TSS wraz ze współczynnikiem SAD nie wykrywa wszystkich poruszających się obiektów.

Algorytm BBGDS wykrył więcej ruchomych bloków niż algorytm TSS. Lecz ocena wizualna działania tego algorytmu jest o wiele gorsza niż wspomnianego TSS. Algorytm ten niepoprawnie wykrył o wiele więcej ruchomych bloków niż TSS. Bez wątpienia wyniki uzyskane przez ten algorytm są najgorsze spośród przebadanych algorytmów.

Wyniki uzyskane przez algorytm 4SS są lepsze niż przez wyniki algorytmu BBGDS ale gorsze niż przez TSS. Algorytm wykrył porównywalną liczbę ruchomych bloków z liczbą bloków wykrytych przez TSS. Ale duża część z pośród wykrytych ruchomych bloków została rozpoznana błędnie.

Trzy wyżej omówione algorytmy uzyskały najsłabsze wyniki spośród pięciu przebadanych algorytmów. Moim zdaniem wyniki uzyskane przez CS i LOGS wykluczają stosowanie tych algorytmów na korzyść wspomnianych CS i LOGS.

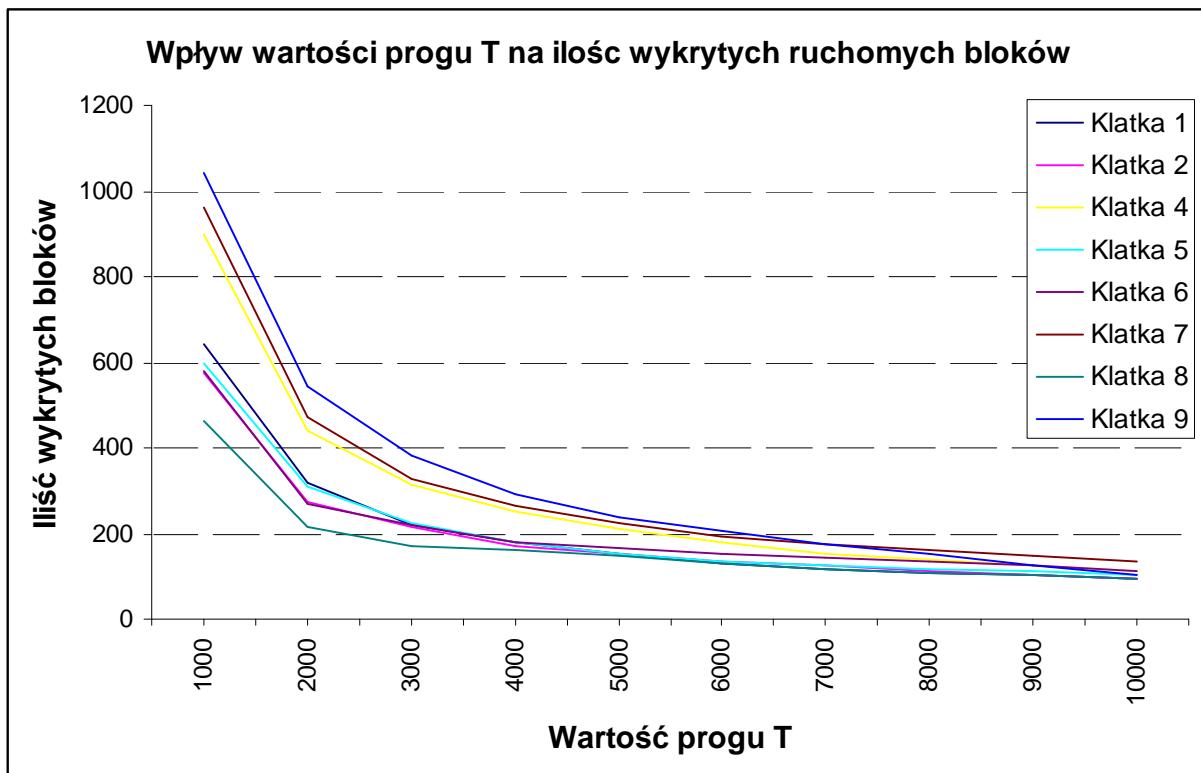
Wyniki badań algorytmów CS i LOGS były identyczne jeśli chodzi o ilość wykrywanych ruchomych bloków. Lecz ocena wizualna działania tych dwóch algorytmów jest już inna. Co prawda algorytmy wykrywają taką samą ilość

ruchomych bloków ale już położenie bloków nie jest już identyczne. Oczywiście spora część z bloków wykrytych przez te algorytmy ma to samo położenie dla obu metod ale jest też część bloków znajdujących się w różnym położeniu. Oba algorytmy mają zadowalające wyniki działania – wykrywają większość ruchomych obiektów. Dlatego też w dalszych testach będę brał pod uwagę tylko te dwa algorytmy.

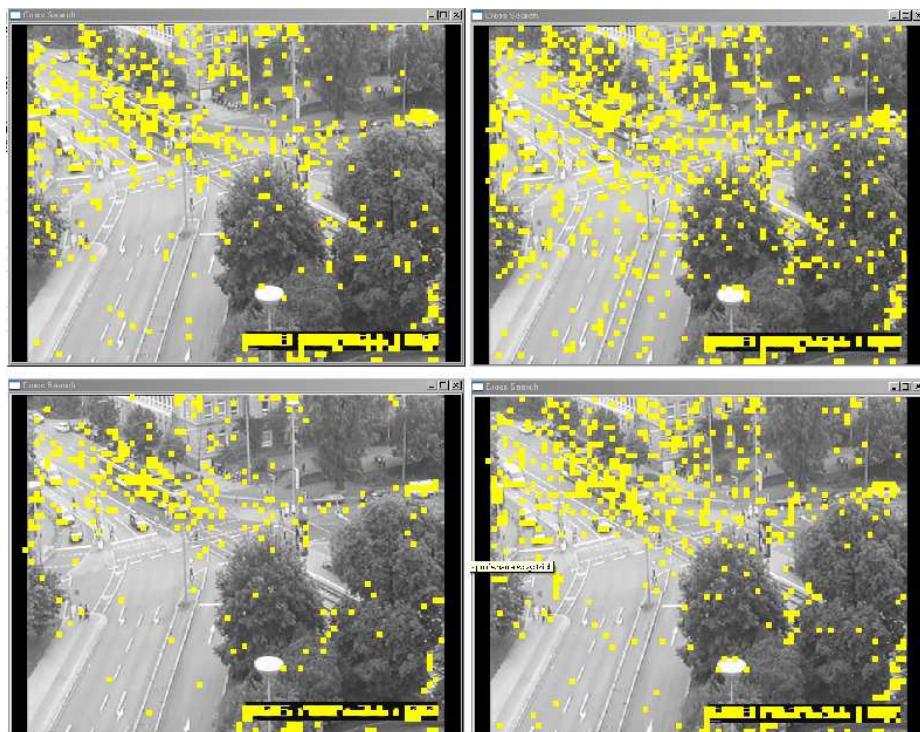
Kolejny test dotyczy wpływu wartości progu T na wyniki osiągane przez algorytm. Jak już wspomniałem w tym teście biorą udział algorytmy CS i LOGS. Oba z algorytmów stosują jako współczynnik dopasowania współczynnik MSD. W poniższej tabeli znajdują się wyniki uzyskane przez te algorytmy a na rysunku widać wizualne wyniki osiągane przez dany algorytm. Jak już wspomniałem oba algorytmy wykrywają taką samą liczbę ruchomych bloków. Na poniższych rysunkach można zaobserwować wpływ różnych parametrów na działanie algorytmów wykrywania ruchu za pomocą metod dopasowywania bloków.

<b>PRÓG (MSD)</b>	<b>Liczba ruchomych bloków wykrywanych w poszczególnych klatkach</b>								
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
<b>1000</b>	644	576	901	354	597	582	962	462	1043
<b>2000</b>	318	276	441	201	309	271	472	214	546
<b>3000</b>	221	214	314	169	224	219	329	172	383
<b>4000</b>	180	173	250	153	181	182	264	160	290
<b>5000</b>	152	155	212	139	155	166	223	149	237
<b>6000</b>	130	136	179	128	135	153	193	130	205
<b>7000</b>	116	125	153	122	124	143	176	119	174
<b>8000</b>	109	113	139	114	118	135	163	110	152
<b>9000</b>	103	103	125	104	111	125	150	104	126
<b>10000</b>	93	94	113	99	102	112	137	94	105

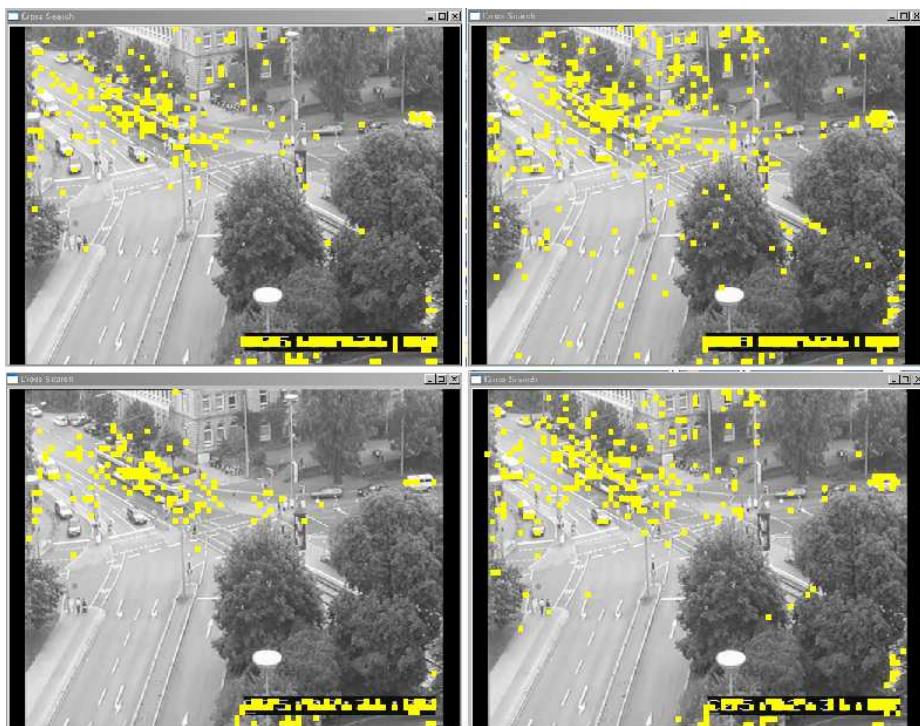
**Tabela 2.** Wpływ wartości progu T na liczbę wykrywanych przez algorytm ruchomych bloków.



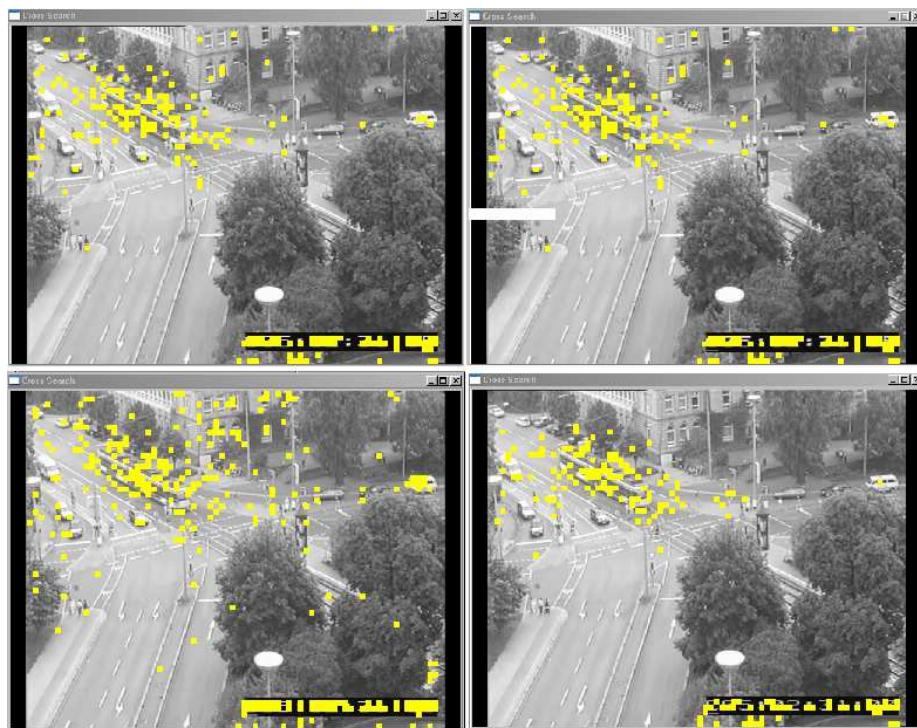
**Rysunek 39.** Wykres przedstawiający wpływ wartości progu na liczbę wykrywanych ruchomych bloków przez algorytm.



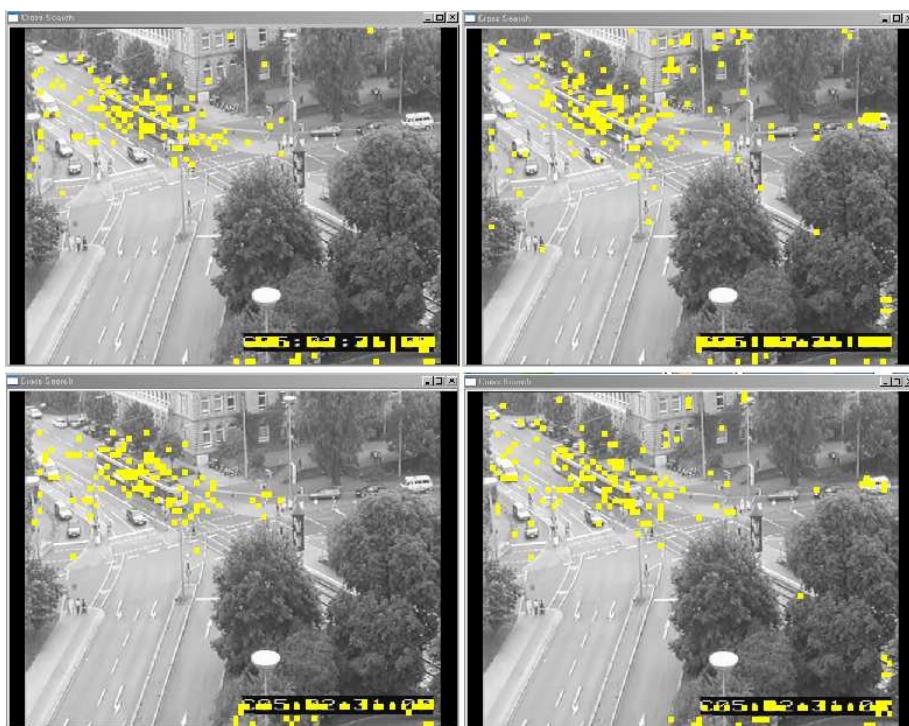
Rysunek 40. Wyniki działania algorytmu CS z progiem wynoszącym 1000.



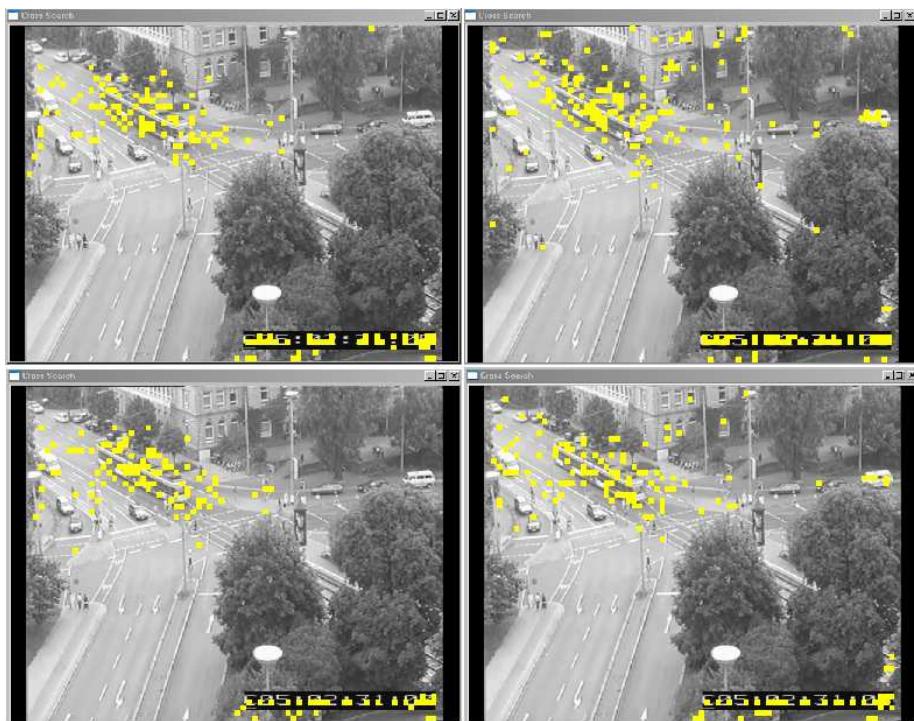
Rysunek 41. Wyniki działania algorytmu CS z progiem wynoszącym 2000.



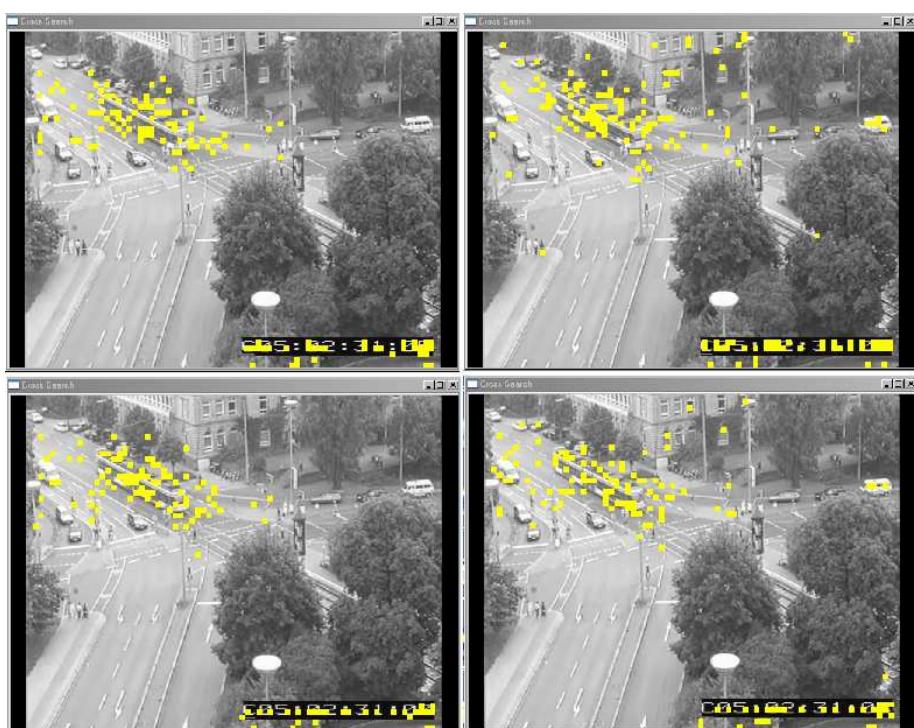
Rysunek 42. Wyniki działania algorytmu CS z progiem wynoszącym 3000.



Rysunek 43. Wyniki działania algorytmu CS z progiem wynoszącym 4000.



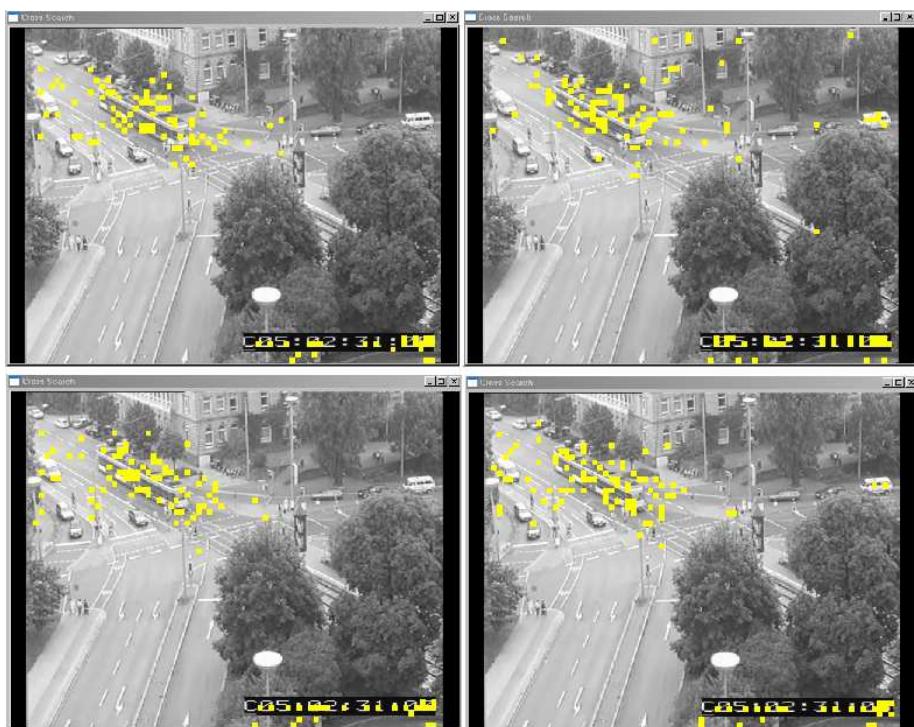
Rysunek 44. Wyniki działania algorytmu CS z progiem wynoszącym 5000.



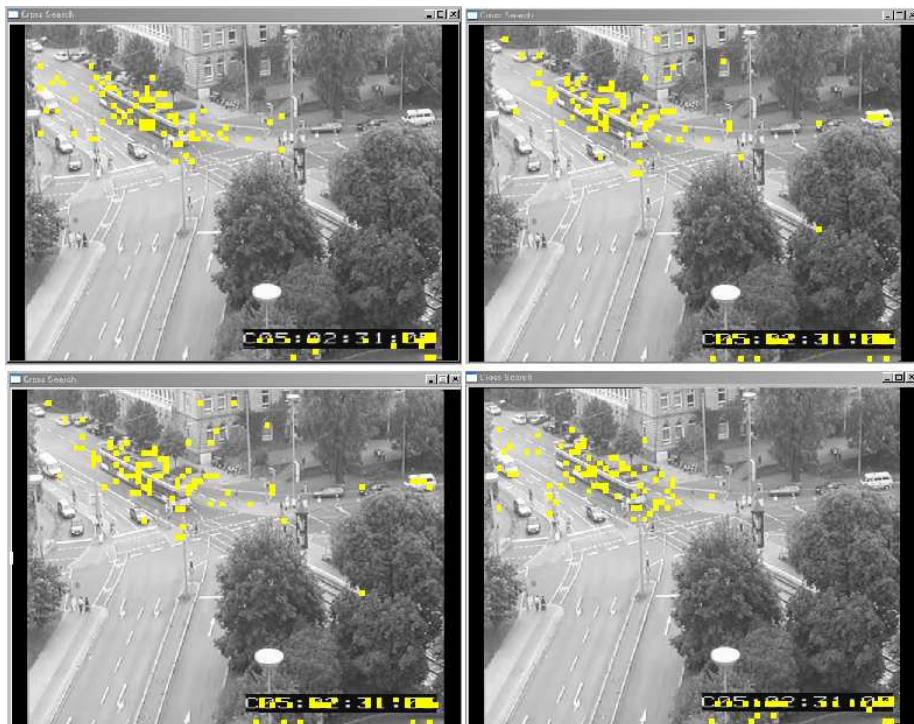
Rysunek 45. Wyniki działania algorytmu CS z progiem wynoszącym 6000.



Rysunek 46. Wyniki działania algorytmu CS z progiem wynoszącym 7000.



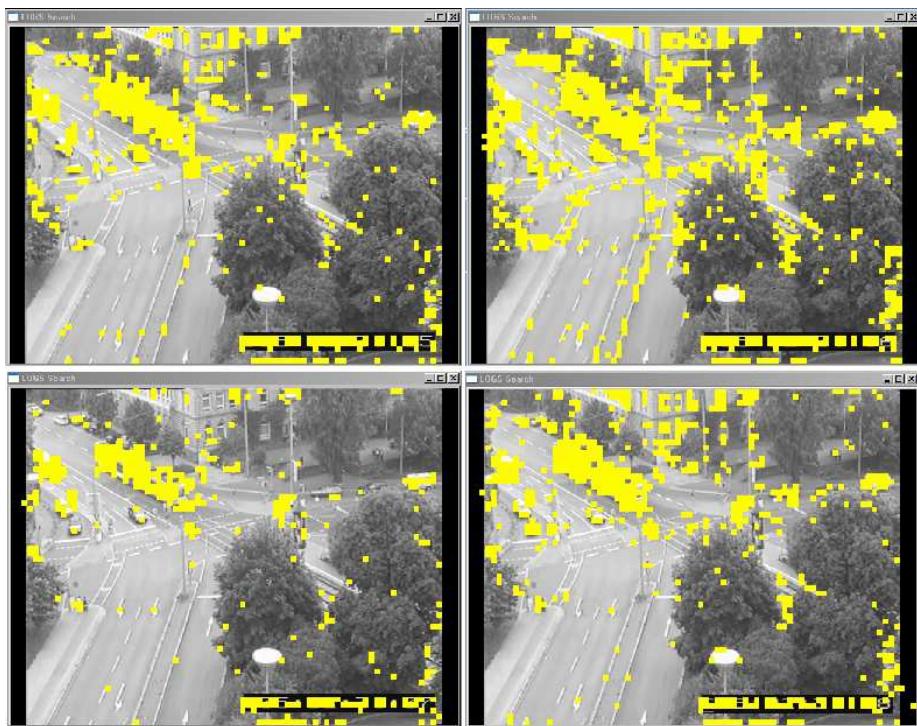
Rysunek 47. Wyniki działania algorytmu CS z progiem wynoszącym 8000.



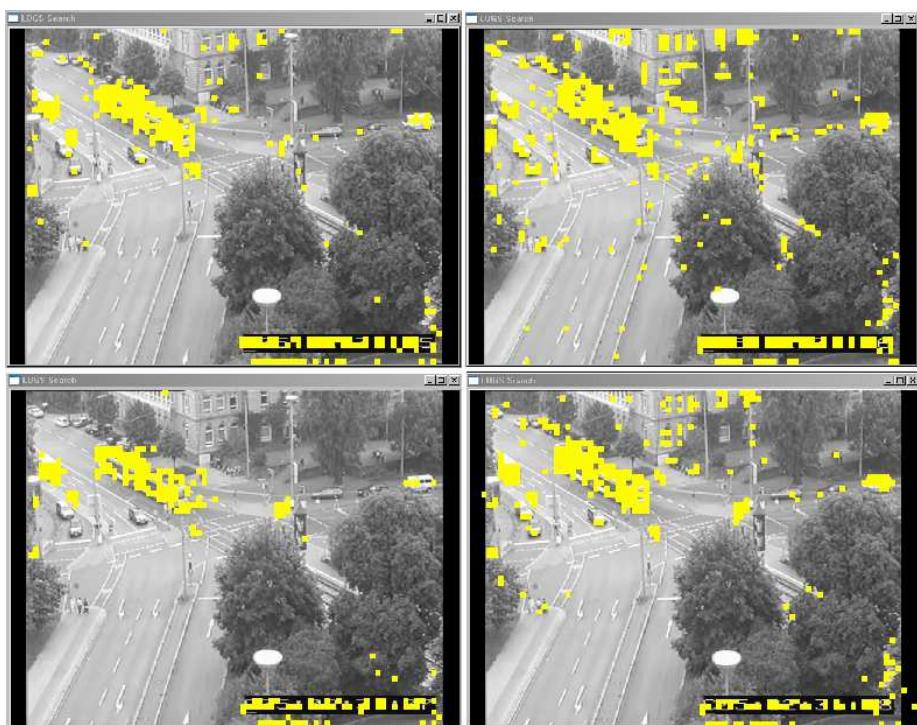
Rysunek 48. Wyniki działania algorytmu CS z progiem wynoszącym 9000.



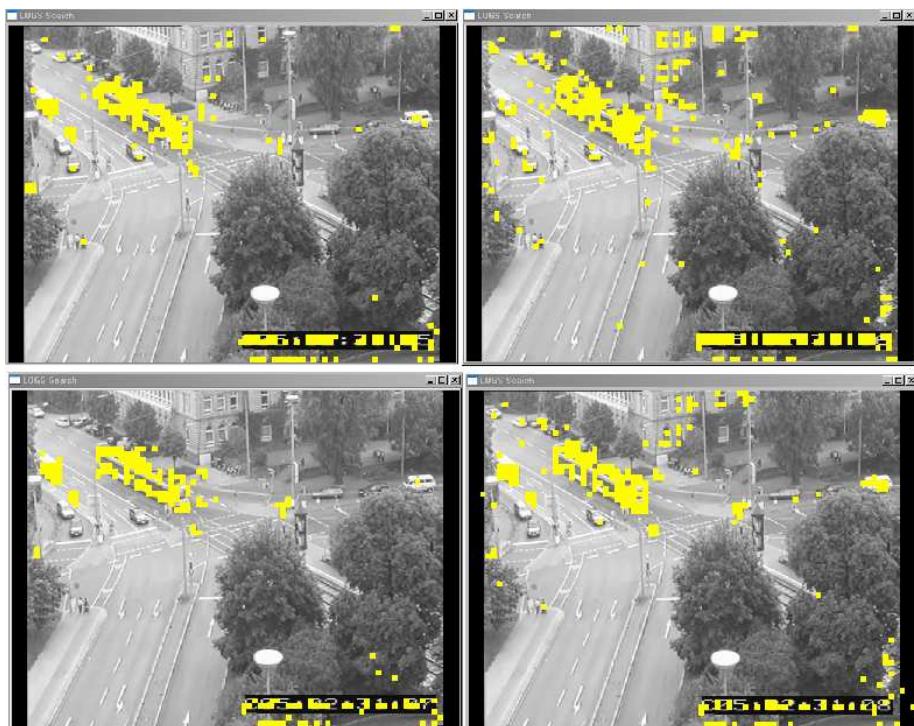
Rysunek 49. Wyniki działania algorytmu CS z progiem wynoszącym 10000.



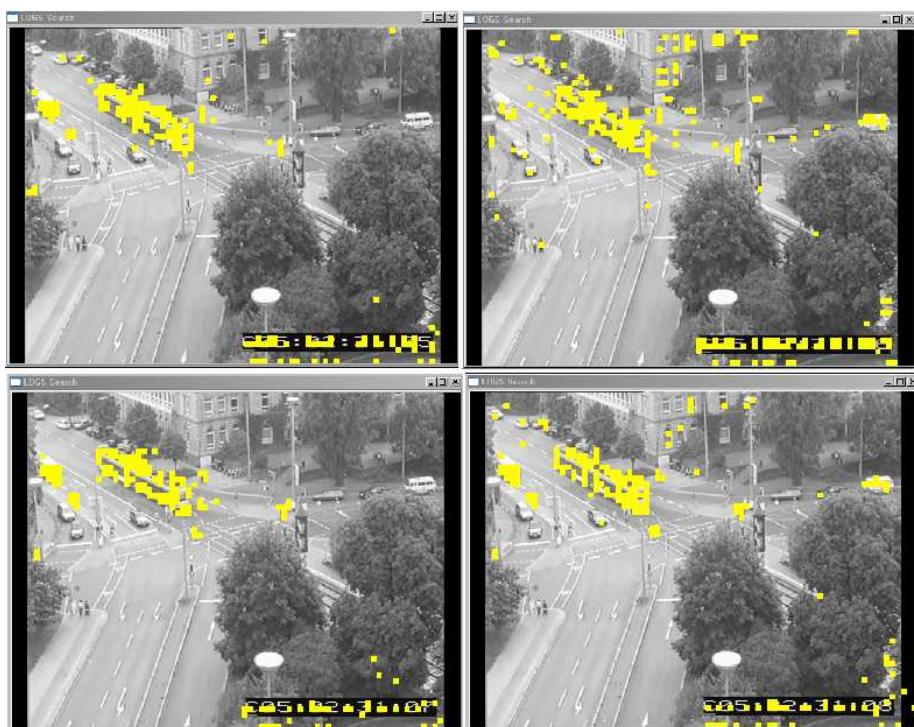
Rysunek 50. Wyniki działania algorytmu LOGS z progiem wynoszącym 1000.



Rysunek 51. Wyniki działania algorytmu LOGS z progiem wynoszącym 2000.



Rysunek 52. Wyniki działania algorytmu LOGS z progiem wynoszącym 3000.



Rysunek 53. Wyniki działania algorytmu LOGS z progiem wynoszącym 4000.



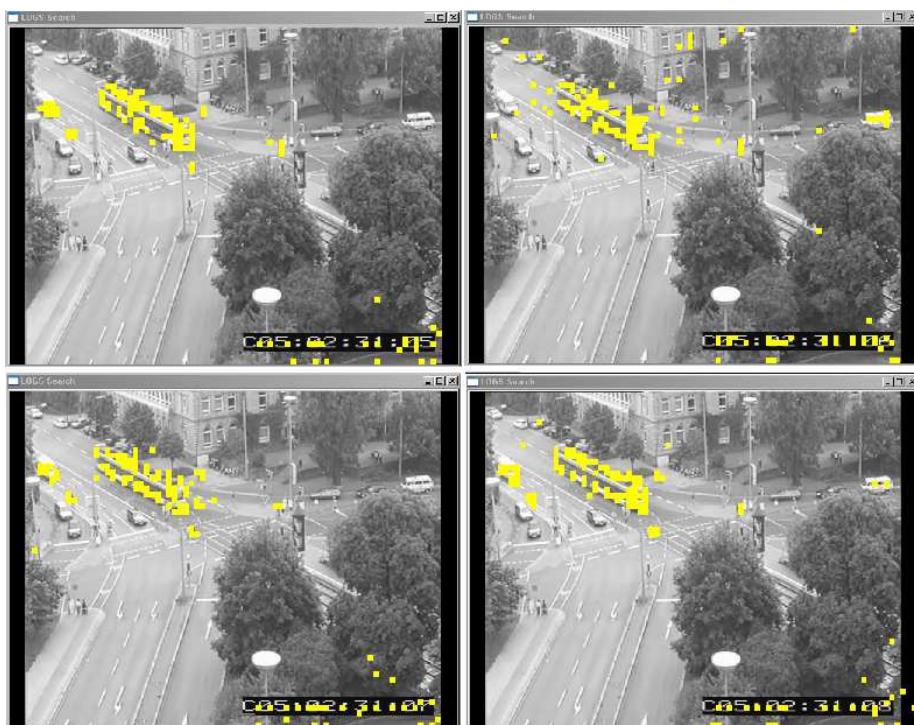
Rysunek 54. Wyniki działania algorytmu LOGS z progiem wynoszącym 5000.



Rysunek 55. Wyniki działania algorytmu LOGS z progiem wynoszącym 6000.



Rysunek 56. Wyniki działania algorytmu LOGS z progiem wynoszącym 7000.



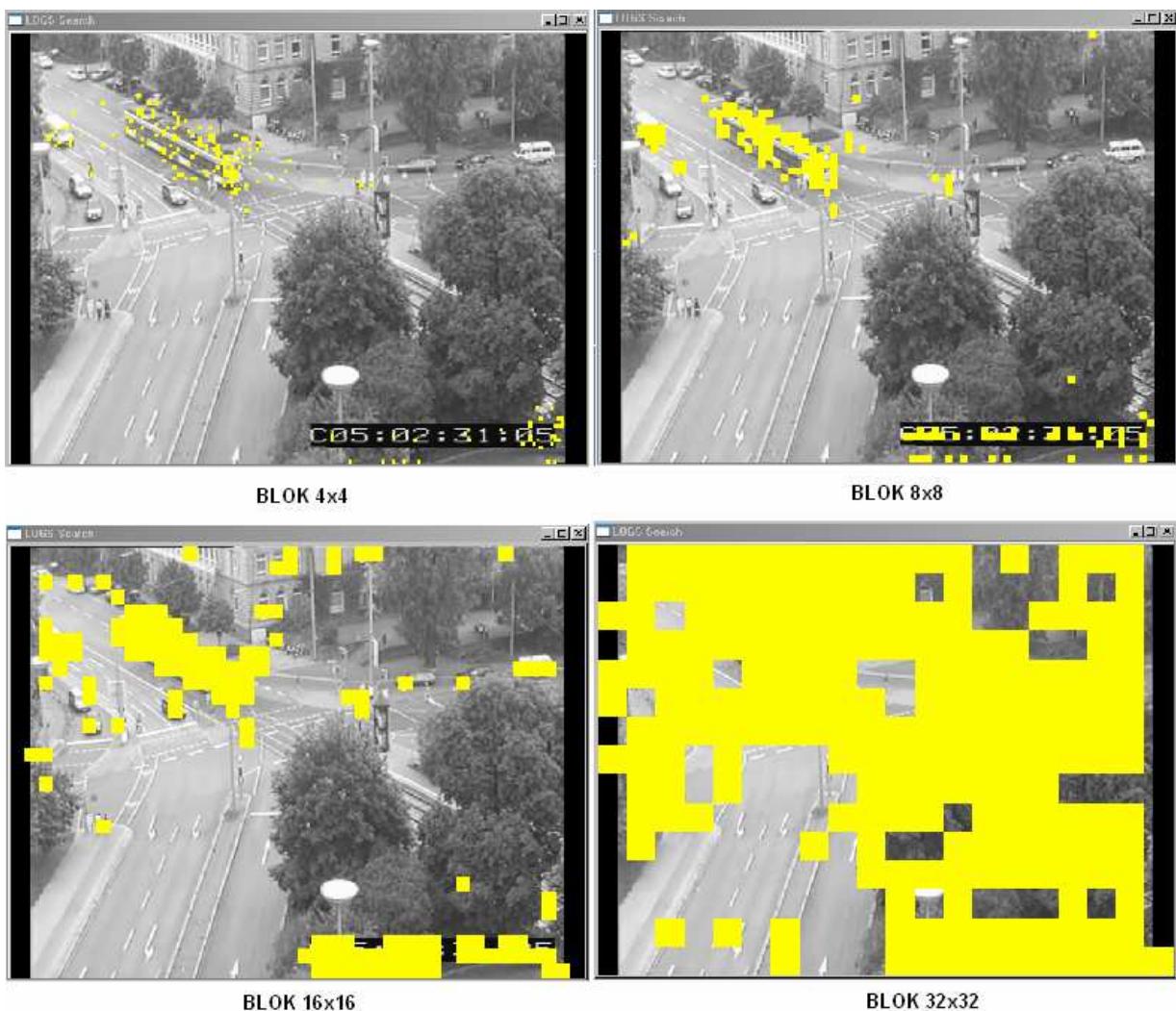
Rysunek 57. Wyniki działania algorytmu LOGS z progiem wynoszącym 8000.



Rysunek 58. Wyniki działania algorytmu LOGS z progiem wynoszącym 9000.



Rysunek 59. Wyniki działania algorytmu LOGS z progiem wynoszącym 10000.



**Rysunek 60.** Wpływ rozmiaru bloku na wyniki działania algorytmu LOGS dla progu  $T$  wynoszącego 5000.

Z powyższych rysunków widać, że najlepsze wyniki osiąga algorytm LOGS. Wykrywa on wszystkie ruchome obiekty.

## 6. Wnioski i podsumowanie

Po przeprowadzeniu testów stwierdzam, że najlepszym algorytmów z pośród tych, które zaimplementowałem jest algorytm LOGS. Wykrywa on wszystkie poruszające się obiekty i jest dość odporny na szумy. Jak można zobaczyć na rysunku wpływ wielkości bloku na działanie algorytmu jest dość znaczny. Dla badanych klatek filmu (640x480 pikseli) wielkość bloku wynosząca 32x32 piksele powoduje nieczytelne działanie algorytmu. Uważam, że najlepsze efekty wizualne algorytm osiąga dla wielkości bloku wynoszącego 16x16 pikseli i dla progu  $T = 4000$ . Należy jednak zwrócić uwagę na wielkość obiektów w badanym filmie. Obiekty te nie są duże w porównaniu z wielkością klatki filmowej. Lecz gdyby obiekty były odpowiednio duże sądę, że można zastosować wielkości bloku wynoszącą 32x32 piksele.

Po przeprowadzeniu testów stwierdzam, że algorytmy wykrywania ruchu za pomocą metod dopasowywania bloków (większość z nich) w mniejszym lub większym stopniu dobrze spełniają swoje zadania. Wykrywają poruszające się obiekty i ułatwiają obserwacje danego obszaru.

Przeprowadzone testy uwydatniają wpływ szumów na algorytmy TSS, BBGDS i 4SS. Bez wątpienia najsłabsze wyniki osiąga algorytm BBGDS.

W pracy przedstawiłem algorytmy wykrywania ruchu za pomocą metod dopasowywania bloków. Opisałem kilkanaście różnych algorytmów i kilka współczynników dopasowania. Znajdująca się tu część teoretyczna jest wystarczająca do tego aby zrozumieć działanie tego typu algorytmów. Przeprowadzone testy pozwalają stwierdzić, że można z powodzeniem stosować tego typu metody do wykrywania ruchu.

## 7. Bibliografia

- [1] **N. S. Love, C. Kamath**, “*An empirical study of block matching techniques for the detection of moving objects.*”, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Styczeń 2006.
- [2] **A. Gyaourova, C. Kamath, S.-C. Cheung**, “*Block matching for object tracking.*”, Center for Applied and Scientific Computing, Lawrence Livermore National Laboratory, Październik 2003.
- [3] **J. Watkinson**, “*The MPEG Handbook, Second Edition.*”, Butterworth-Heinemann, str. 146 - 153, 2001.
- [4] **M. K. Stelios, G. R. Martin, R. A. Packwood**, “*Parallelisation of Block Matching Motion Estimation Algorithms.*”, Technical Report CS-RR-320, Department of Computer Science, University of Warwick, Coventry, UK, 1 1997.
- [5] **H. Jozawa, K. Kamikura**, “*Two-stage motion compensation using adaptive global MC and local affine MC.*”, IEEE Trans. Circuits Syst. Video Technol. vol. 7, no. 1 Styczeń 1997.
- [6] **C. –L. Huang, C. –Y. Hsu**, “*A new motion compensation method for image sequence coding using hierarchical grid interpolation.*”, IEEE Trans. on Circuits and Systems for Video Technol. vol. 4, no. 1, strony 42 - 52, Luty 1994.
- [7] **Y. Nie, K. –K. Ma**, “*Adaptive rood pattern search for fast block-matching motion estimation.*”, IEEE Trans. on Image Process., vol. 11, no. 12, Grudzień 2002, strony 1442 - 1449.
- [8] **X. Lee, Y. –Q. Zhang**, “*A fast hierarchical motion-compensation scheme for video coding using block feature matching.*”, IEEE Trans. on Circuits and

Systems for Video Techn., vol. 6, no. 6, Grudzień 1996, strony 627 - 635.

- [9] **J. R. Jain, A. K. Jain**, “*Displacement measurement and its application in interframe image coding.*”, IEEE Trans. Commun., COM-29: 1799 - 1808, 1981.
- [10] **T. Koya, K. Iinuma, A. Hirano, Y. Iiyima**, and T. Ishiguro, “*Motion-compensated interframe coding for video conferencin.*”, in Proc. NTC 81, strony G5.3.1 - G5.3.5, New Orleans, LA, Grudzień 1981.
- [11] **L. M. Po, W. C. Ma**, “*A Novel Four-Step Search Algorithm for Fast Block Motion Estimation.*”, IEEE Trans. Circuits Syst. Video Technol., vol. 6, No. 3, strony 313 - 317, Czerwiec 1996.
- [12] **L. -K. Liu, E. Feig**, “*A block-based gradient descent search algorithm for block motion estimation in video coding.*”, IEEE Trans. on Circuits and Systems for Video Techn., vol. 6, no. 4, Sierpień 1996, strony 419 - 422.
- [13] **S. Zhu, K. -K. Ma**, “*A new diamond search algorithm for fast block-matching motion estimation.*”, Int. Conf. on Inform., Commun. and Signal Process. ICICS'97, Singapur, Wrzesień 1997, strony 292 - 296.
- [14] **C. Zhu, X. Lin, L. -P. Chau, K. -P. Lim, H. -A. Ang, C. -Y. Ong**, “*A novel hexagon-based search algorithm for fast block motion estimation.*”, Centre for Signal Processing, School of Electrical & Electronic Engineering Nanyang Technological University, Singapur.
- [15] **R. Li, B. Zeng, M. L. Liou**, “*A new three-step search algorithm for block motion estimation.*”, IEEE Trans. on Circuits and Systems for Video Techn., vol. 4, no. 4, Sierpień 1994, strony 438 - 442.

- [16] **L. Luo, C. Zou, X. Gao**, "A new prediction search algorithm for block motion estimation in video coding.", IEEE Trans. on Cons. Electronics, vol. 43, no. 1, Luty 1997, strony 56 - 61.
- [17] **B. Jahne**, "Digital Image Processing 5<sup>th</sup> revised and extended edition.", Springer-Verlag, strony 375 - 412, 2002.
- [18] **E. L. McHugh**, "Video motion detection for real-time hazard warnings in surface mines.", Spokane Research Laboratory, National Institute for Occupational Safety and Health, Spokane, Washington, USA.
- [19] **B. K. P. Horn, B. G. Schunck**, "Determining optical flow.", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, strony 185 - 203.
- [20] **H. Hausecker, H. Spies**, "Computer Vision and Applications - A Guide for Students and Practitioners.", Academic Press, A Harcourt Science and Technology Company, strony 347 - 392, 2000.
- [21] **B. Lucas, T. Kanade**, "An iterative image registration technique with an application to stereo vision.", DARPA Image Understanding Workshop, strony 121 - 130.
- [22] **J. K. Kearney, W. B. Thompson, D. L. Boley**, "Optical flow estimation: an error analysis of gradient-based methods with local optimization.", IEEE Trans. PAMI 9(2), strony 229 - 249.
- [23] **E. H. Adelson, J. R. Bergen**, "The extraction of spatiotemporal energy in human and machine vision.", Proc. IEEE Workshop Visual Motion, strony 151 - 156.

- [24] **E. P. Simoncelli, E. H. Adelson, D. J. Heeger**, “*Probability distributions of optical flow.*”, Proc. Conf. Comput. Vis. Patt. Recog., strony 310 - 315, Mani 1991.
- [25] **W. H. Press, S. A. Tenkolsky, W. Vetterling, B. Flannery**, “*Numerical Recipes in C: The Art of Scientific Computing.*”, New York: Cambridge Univeristy Press, 1992.
- [26] **H. Nagel**, “*Displacement vectors derived from second-order intensity variations in image sequences.*”, Computer Vision, Graphics and Image Processing, 21: 85 - 117, 1983.
- [27] **O. Tretiak, L. Pastor**, “*Velocity estimation from image sequences with second order differential operators.*”, Proc. 7<sup>th</sup> Intern. Conf. Patt. Recogn., Montreal 1994, strony 20 - 22.
- [28] **S. Uras, F. Girosi, A. Verri, V. Torre**, “*A computational approach to motion perception.*”, Biol. Cybern., 60: 79 - 97, 1988.
- [29] **M. J. Black, P. Anandan**, “*The robust estimation of multiple motions: parametric and piecewise-smooth flow fields.*”, Computer Vision and Image Understanding, 63(1): 75 - 104, 1996.
- [30] **B. K. Horn, B. G. Schunk**, “*Determining optical flow.*”, Artificial Intelligence, 17: 185 - 204, 1981.
- [31] **A. Bainbridge-Smith, R. G. Lane**, “*Determining optical flow using a differential method*”, Image and Vision Computing, 15: 11 - 22, 1997.

- [32] **H. Nagel**, “*Image sequences - ten (octal) years – from phenomenology towards a theoretical foundation.*”, Proc. Int. Conf. Patt. Recogn., Paris 1986, strony 1174 - 1185, Washington: IEEE Computer Society Press, 1986.
- [33] **H. Nagel**, “*On the estimation of optical flow: relations between different approaches and some new results.*”, Artificial Intelligence, 33: 299 - 324, 1987.
- [34] **J. Bigun, G. H. Granlund**, “*Optimal orientation detection of linear symmetry.*”, Proceedings ICCV'87, London 1987, strony 433 - 438, IEEE Washington, DC: IEEE Computer Society Press.
- [35] **M. Kass, A. Witkin**, “*Analyzing oriented patterns.*”, Comp. Vision Graphics Image Proc., 37: 362 - 385, 1987.
- [36] **H. Knutsson**, “*Filtering and Reconstruction in Image Processing.*”, Diss. Linkoping Univ., 1982.
- [37] **H. Knutsson**, “*Representing local structure using tensors.*”, Proc. 6<sup>th</sup> Skandinavian Conf. on Image Analysis, Oulu, Finland, strony 244 - 251, Springer-Verlog, 1998.
- [38] **G. H. Granlund, H. Knutsson**, “*Signal Processing for Computer Vision.*”, Kluwer, 1995.
- [39] **B. Jahne**, “*Digital Image Processing-Concepts, Algorithms, and Scientific Applications 4<sup>th</sup>, edition.*”, New York: Springer, 1997.
- [40] **D. J. Heeger**, “*Optical flow from spatiotemporal filters*”, Int. J. comp. Vis., 1: 279 - 302, 1988.

- [41] **D. J. Heeger**, "Model for the extraction of image flow.", *J. Opt. Soc. Am. A.*, 4: 1455 - 1471, 1987.
- [42] **Y. -K. Wang, G. -F. Tu**, "Fast Binary Block Matching Motion Estimation using Efficient One-Bit Transform.", Department of Electrical Engineering, the Graduate School of Academia Sinica, Beijing, Chiny.
- [43] **X. Lee**, "A fast feature matching algorithm of motion compensation for hierarchical video codec.", Proc. SPIE VCIP-92, Boston, MA, USA, 1992, vol. 18818, strony 1462 - 1474.
- [44] **J. Feng, K. -T. Lo, H. Mehrpour, A. E. Karbowiak**, "Adaptive block matching motion estimation algorithm using bit-plane matching.", Proc. ICIP-95, Washington DC, USA, strony 496 - 499, 1995.
- [45] **B. Natarajan, V. Bhaskaran, K. Konstantinides**, "Low-complexity block-based motion estimation via one-bit transforms.", *IEEE Trans. Circuits Syst. Video Technol.* vol. 7, no. 4, strony 702 - 706, Sierpień 1997.
- [46] **J. Feng, K. -T. Lo, H. Mehrpour, A. E. Karbowiak**, "Adaptive block matching algorithm for video compression.", *IEEE Proc. Vis. Image Signal Processing*, vol. 145, no. 3, strony 173 - 178, Czerwiec 1998.
- [47] **M. M. Mizuki, U. Y. Desai, I. Masaki, A. Chandrakasan**, "A binary block matching architecture with reduced power consumption and silicon area requirement.", Proc. IEEE ICASSP-96, Atlanta, USA, 1996, vol. 6, strony 3248 - 3251.
- [48] **Y. -L. Chan, W. -C. Sin**, "Edge oriented block motion estimation for video coding.", *IEEE Proc. Vis. Image Signal Processing*, vol. 144, no. 3, strony 136 - 144, Czerwiec 1997.

- [49] **P. H. W. Wong, O. C. An**, “*Modified one-bit transform for motion estimation.*”, IEEE Trans. Circuits Syst. Video Technol., vol. 9, no. 7, strony 1020 - 1024, Październik 1999.
- [50] **A. Puri, H. –M. Hong, D. L. Schilling**, “*An efficient block-matching algorithm for motion compensated coding.*”, Proc. IEEE CASSP-87, strony 1063-1066.
- [51] **C. –H. Hsieh, P. –C Lu, J. –S. Shyn, E. –H. Lu**, “*Motion estimation algorithm using interblock correlation.*”, Electron. Lett., vol. 26, no. 5, strony 276 - 277, Marzec 1990.
- [52] **S. Zefar, Y. –Q. Zhang, J. S. Baras**, “*Predictive block-matching motion estimation for TV coding – part I: inter-block prediction.*”, IEEE Trans. Broadcasting, vol. 37, no. 3, strony 97 - 101, Wrzesień 1991.
- [53] **Y. –Q. Zhang, S. Zafar**, “*Predictive block-matching motion estimation for TV coding – part II: inter-frame prediction.*”, IEEE Trans. Broadcasting, vol. 37, no. 3, strony 97 - 102, Wrzesień 1991.
- [54] **R. Brad, I. A. Letia**, ”*Cloud Motion Detection from Infrared Satellite Images.*”, University of Sibiu Computer Science Department Bulevardul Victoriei.
- [55] **L. Di Stefano, E. Viarani**, ”*Vehicle Detection and Tracking Using the Block Matching Algorithm.*”, Department of Electronics, Computer Science and Systems (DEIS) University of Bologna.
- [56] **T. Shinohara**, ”*Differential motion detection method using background image.*”, United States Patent, numer 5,606,376, 25 luty 1997.

- [57] **E. Sugimoto, T. Urano, S. Kobayashi, Y. Hamamoto, H. Kodama,** "Motion video coding systems with motion vector detection.", United States Patent, numer 5,808,700, 15 wrzesień 1998.
- [58] **J. Konrad,** "Motion detection and estimation.", Image and Video Processing Handbook, roz. 3.8, Academic Press, 2000.
- [59] **A. Jencik, C. v. Planta, J. Conradt,** "Motion Detection in the Visual System of the Fly.".
- [60] **J. Kennedy, R. Eberhart,** "Particle swarm optimization.", Proc. IEEE Int. Conf. on Neural Networks, 1995: 1942-1948.
- [61] **R. Eberhart, J. Kennedy,** "A new optimize using particle swarm theory.", Proc. 6<sup>th</sup> Int. Symposium on Micro Machine and Human Science, 1995: 39-43.
- [62] **R. Ren, M. Manokar, Y. Shi, B. Zheng,** "A Fast Block Matching Algorithm for Video Motion Estimation Based on Particle Swarm Optimization and Motion Prejudgment.", School of Communications and Information Engineering, Nanjing University of Posts and Telecommunications.
- [63] **I. Stuke, T. Aach, E. Barth, C. Mota,** "Multiple-Motion-Estimation by Block-matching using MRF.", International Journal of Computer & Information Science, Vol. 5, No. 1, Marzec 2004.
- [64] **L. Luo, C. Zou, X. Gao, Z. He,** "A new prediction search algorithm for block motion estimation in video coding.", IEEE Transactions on Consumer Electronics, Vol. 43, No. 1, Luty 1997, strony 56 – 61.
- [65] **H. Fan, C. Zhang,** "Some improvements on MPEG2 video coding algorithm.", J. China Institute of communications, vol. 17, no. 5, Maj 1996.

- [66] **C. -H. Lee, L. -H. Chen**, "A Fast Motion Estimation Algorithm Based on the Block Sum Pyramid.", IEEE Trans. on Image Process., vol. 6, no. 11, Listopad 1997, strony 1587 -1591.
- [67] **Y. Yue, Z. Jian, W. Yiliang, L. Fengting, G. Chenghui**, "A fast effective block motion estimation algorithm", Proceedings of ICSP'98, strony 827 – 830.
- [68] **D. Liu, W. Sun**, "Block-based fast motion estimation algorithms in video compression", Department of Electrical and Computer Engineering, Stereus Institute of Technology, Hoboken, Sierpień 1998.
- [69] **S. C. Kwatra, C. -M. Liu, W. A. Whyte**, "An adaptive algorithm for motion compensated color image coding", IEEE Trans. of Commun., vol. COM-35, strony 747 – 753, Czerwiec 1987.
- [70] **J. P. Berns, T. G. Noll**, "A flexible and cascadable 100 GOPS variable-size block-matching motion estimation emulator chip for HDTV applications", Chair on Electrical engineering and Computer Systems, University of Technology RWTH, Aachen, Niemcy.
- [71] **R. Srinivasan, K. R. Rao**, "Predictive coding based on efficient motion estimation", IEEE Trans. on Commun., vol. COM-33, strony 888 – 895, sierpień 1985.
- [72] **S. L. Kilthau, M. S. Drew, T. Moller**, "Full search content independent block matching based on the fast Fourier transform", IEEE ICIP'2002, strony 669 – 672.
- [73] **J. -B. Xu, L. -M. Po, C. -K. Cheung**, "Adaptive motion tracking block matching algorithm for video coding", IEEE Trans. On Circuits And Systems For Video Technology, vol. 9, no. 7, Październik 1999, strony 1025 – 1027.

- [74] **H. Y. Chung, P. Y. S. Cheung, N. H. C. Yung**, “*Adaptive search center non-linear Three Step Search*”, Department of Electrical&Electronic Engineering, The University of Hong Kong.
- [75] **D. Gui-guang, G. Bao-long**, “*Motion vector estimation using line-square search block matching algorithm for video sequences*”, EURASIP Journal on Applied Signal Processing, Listopad 2004, strony 1750 – 1756.
- [76] **D. Bagni, P. Zoratti**, “*Block matching for automotive applications on Spartan-3A DSP devices*”, Xcell Journal, pierwszy kwartał 2008, strony 16 – 19.
- [77] **D. Moore**, “*A real-world system for human motion detection and tracking*”, California Institute of Technology, Czerwiec 2003.
- [78] **J. Ruiz-del-Solar, P. A. Vellejos**, “*Motion detection and object tracking for an AIBO Robot Soccer Player*”, Robotic Soccer, Pedro Lima, Itech Education and Publishing, Wiedeń, Austria, Grudzień 2007, strony 337 – 346.
- [79] **W. von Seelen, C. Curio, J. Gayko, U. Handmann, T. Kalinke**, “*Scene analysis and organization of behaviour in driver assistance systems*”, IEEE Image Processing, vol. 3, 2000, strony 524 – 527.
- [80] **T. Schafer, P. Wasmeier, K. Ratke, K. Foppe, G. Preuss**, “*Motion detection at Munich’s Olympic Tower with a multi-sensor system operating at different sampling rates*”, Shaping the Change, XXIII FIG Congress, Monachium, Październik 2006.