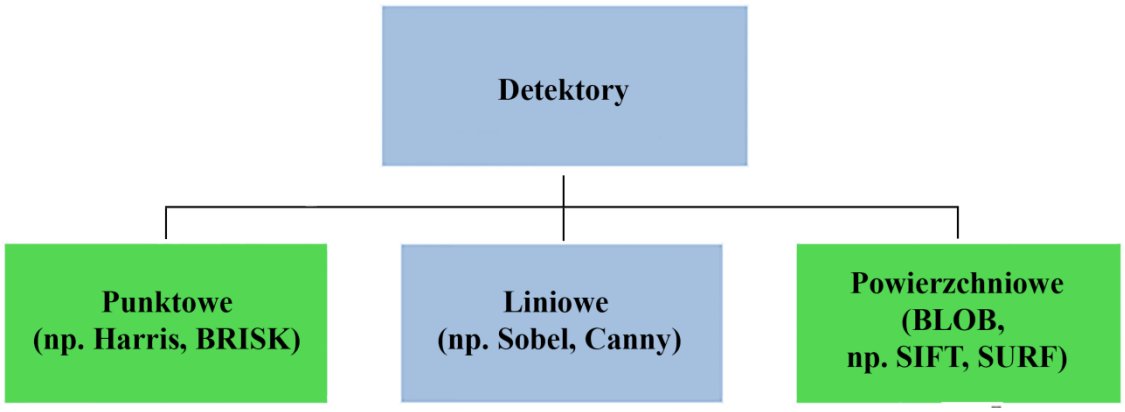
1. Algorytmy detekcji punktów/obszarów na zdjęciach.

Większość obecnie stosowanych programów do orientacji zdjęć naziemnych wykorzystuje algorytmy, które analizują obszar całego zdjęcia, na którym to dopasowywane są do siebie jedynie wybrane fragmenty, charakteryzujące się zbliżonymi cechami. Przyjmuje się zatem pewne założenia definiujące dany punkt jako punkt charakterystyczny [1]:

- jednoznaczność punktów wykrywanych przez algorytmy FBM – wykrywane punkty muszą odróżniać się od najbliższych sąsiadów. Wskazane jest, aby wykrywane punkty były wysoce kontrastowe, co wyróżniałoby je w lokalnym otoczeniu.
- niezmienność – punkt powinien być wykryty niezależnie od zmian w dystorsji radiometrycznej czy geometrycznej. Warunek ten w największym stopniu wpływa na dokładność, poprawność i niezawodność procesu dopasowywania punktów charakterystycznych.
- stabilność – punkty powinny być wykrywane zawsze jednakowo niezależnie od warunków otoczenia (np. zmian oświetlenia czy skali).
- interpretowalność – podział wykrywanych cech według typu (np. krawędź czy narożnik).
- unikalność – elementy wykrywane na zdjęciach powinny charakteryzować się unikalnością, umożliwiającą ich jednoznaczne wyodrębnienie na podstawie pewnych cech charakterystycznych. Właściwość ta jest szczególnie istotna dla obrazów zawierających powtarzalną teksturę.

Standardowo do wykrywania punktów charakterystycznych na zdjęciach wykorzystywane są trzy rodzaje detektorów tj. punktowe, liniowe oraz powierzchniowe (Rys. 12).



Rysunek 1 Przykładowe rodzaje detektorów punktowych, liniowych i powierzchniowych (BLOB) wykorzystywanych na obrazach cyfrowych

[1] Geetanjali Babbar and others, 'Comparative Study of Image Matching Algorithms', *International Journal of Information Technology and Knowledge Management*, 2.2 (2010), 337–39; Tinne Tuytelaars and Krystian Mikolajczyk, 'Local Invariant Feature Detectors: A Survey', *Foundations and Trends® in Computer Graphics and Vision*, 3.3 (2007), 177–280 <https://doi.org/10.1561/0600000017>; Martin Weinmann, *Visual Features—From Early Concepts to Modern Computer Vision*, *Advances in Computer Vision and Pattern Recognition*, 2013 <https://doi.org/10.1007/978-1-4471-5520-1>.

1.1. Operatory detekcji narożników

W klasycznym ujęciu przetwarzania obrazów w pierwszym etapie przetwarzania danych wykrywane są narożniki (punkty charakterystyczne), których zaletą jest możliwość działania na oryginalnym, surowym, niepoddanym wcześniejszemu przetwarzaniu obrazie. Zastosowanie dodatkowych przekształceń oryginalnego obrazu pozwala zwiększyć efektywność działania algorytmu, co wpływa na większą liczbą poprawnie wykrytych narożników oraz ich lepsze rozmieszczenie.

1.1.1. Detektor Harrisa

Jednym z najstarszych i najczęściej wykorzystywanych detektorów do wykrywania punktów charakterystycznych (tzw. narożników), do którego odwołuje się większość opracowań bazujących w szczególności na wykrywaniu krawędzi, jest detektor Harrisa[2]. Algorytm ten opiera się na trzyetapowym schemacie działania. W pierwszym kroku wykorzystywany jest operator detekcji krawędzi. Następnie definiowany jest próg filtracji, usuwający punkty o zbyt małej wartości operatora. Ostatni etap polega na zastosowaniu supresji punktów nie będących lokalnymi maksimami funkcji. Równocześnie dla każdego niezerowego punktu określany jest okrąg tzw. promień sąsiedztwa, w którym może znajdować się tylko jeden punkt będący lokalnym maksimum funkcji (tj. narożnik).

Zmiany wartości intensywności gradientu dla detektora Harrisa określane są poprzez sumę różnic kwadratów intensywności w badanym oknie poszukiwawczym. Wartość gradientu powinna wskazywać znaczącą zmianę w dwóch kierunkach. Zmiana tylko w jednym kierunku oznacza zazwyczaj jedynie wykrycie pojedynczej krawędzi przez detektor. Zależność tę przedstawia wzór:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

gdzie: $E(u,v)$ – błąd średniokwadratowy pomiędzy fragmentem obrazu przetwarzanego a oknem w, p,$w(x,y)$ – okno przesunięcia, $I\,(x,y)$ – wartość intensywności na zdjęciu, $I\,(x+u, y+v)$ – wartość intensywności uzyskana w oknie przesuniętym o wartość u i v

Jeśli w badanym fragmencie zdjęcia znajduje się narożnik, to funkcja $E(x,y)$ przybiera w tym miejscu wartości maksymalne:

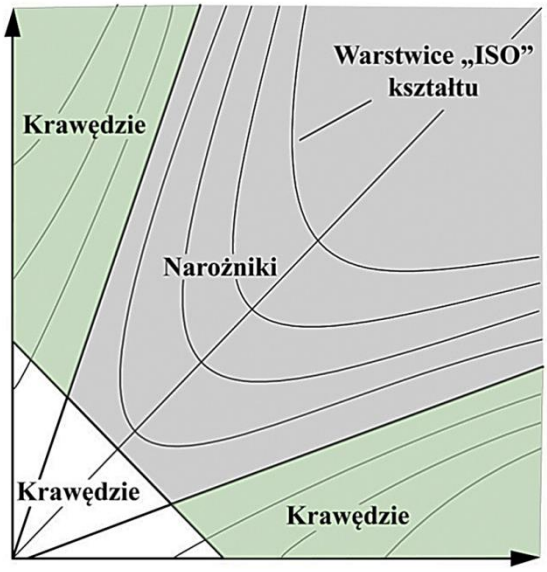
$$E(u,v) \approx [u,v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

gdzie: M – macierz autokorelacji definiowana jako:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Powstała wskutek przekształceń macierz autokorelacji M jest macierzą wartości lokalnych otoczenia badanego punktu. W zależności od przyjętych wartości własnych macierzy definiuje się przynależność punktu do jednej z trzech klas: narożników, krawędzi i ogólnych kształtów (Rys. 13).

[2] (Harris i Stephens, 1988)



Rysunek 2 Przykład interpretacji wartości własnych macierzy autokorelacji M. Punkt zostaje uznany za narożnik w przypadku gdy obie wartości są duże[3]

Analiza wartości własnych macierzy pozwala na klasyfikację punktów wedle następującego klucza:

- jeśli obie wartości własne macierzy są niskie to punkt znajduje się poza krawędzią,
- jeśli obie wartości własne macierzy są wysokie to punkt jest narożnikiem,
- jeśli tylko jedna z wartości macierzy jest wysoka to punkt znajduje się na krawędzi.

Przykład implementacji funkcji Harrisa został przedstawiony przy wykorzystaniu funkcji 1 (wyznaczenie wartości w oparciu o wzór) i 2 (przy wykorzystaniu biblioteki OpenCV):

```
#Import biblioteki OpenCV I NumPy
import cv2
import numpy as np

zdjecie = cv2.imread('moje_zdjecie.png')
szareZdjecie = cv.cvtColor(zdjecie, cv2.COLOR_BGR2GRAY) # Konwersja obrazu do postaci zdjęcia w odcieniach szarości

# Maska filtru kierunkowego Sobel’a (x)
sobelX = np.array((
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]), dtype="int32")

# Maska filtru kierunkowego Sobel’a (y)
sobelY = np.array((
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]), dtype="int32")

# Maska filtru Gaussa o wymiarach 3x3
maskaFiltruGaussa = np.array((
    [1/16, 2/16, 1/16],
```

(1)

[3] (Harris i Stephens, 1988)

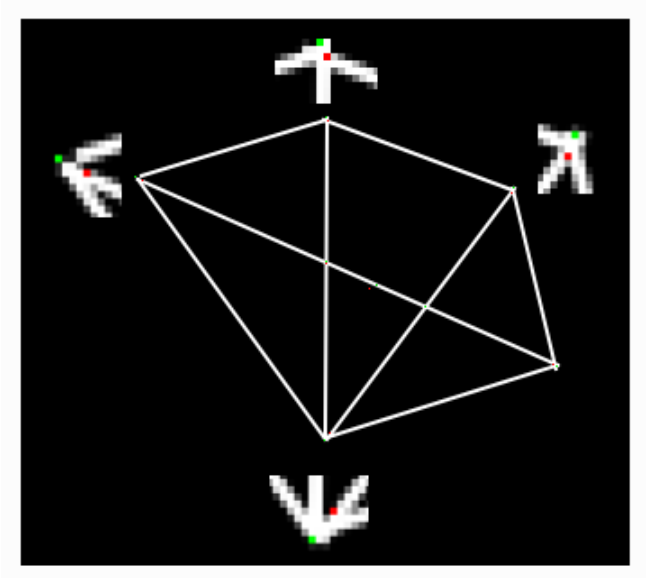
```
    [2/16, 4/16, 2/16],
    [1/16, 2/16, 1/16]), dtype="float64")

threshold=0.09 # Wartość progu, dla którego piksele traktowane są jako punkty

dx = cv2.filter2D(src = szareZdjecie, ddepth = -1, kernel = sobelX)
dy = cv2.filter2D(src = szareZdjecie, ddepth = -1, kernel = sobelY)

dx2 = np.square(dx)
dy2 = np.square(dy)
dxdy = dx*dy

g_dx2 = cv2.filter2D(dx2, -1, maskaFiltruGaussa)
g_dy2 = cv2.filter2D(dy2,-1,  maskaFiltruGaussa)
g_dxdy = cv2.filter2D(dxdy, -1, maskaFiltruGaussa)

harris = g_dx2*g_dy2 - np.square(g_dxdy) - 0.04*np.square(g_dx2 + g_dy2) # r(harris) = det - k*(trace**2)

# Normalizacja wartości w przedziale (0-1)

cv2.normalize(harris, harris, 0, 1, cv2.NORM_MINMAX)

loc = np.where(harris >= threshold)
# Zaznaczenie na obrazie punktów w postaci okręgów
for pt in zip(*loc[::-1]):
   cv2.circle(zdjecie, pt, 1, (0, 0, 255), -1)

cv2.imshow('Punkty wykryte detektorem Harrisa', zdjecie)
if cv2.waitKey(0) & 0xff == 27:
   cv2.destroyAllWindows()
```

```
#Import biblioteki OpenCV I NumPy
import cv2
import numpy as np

zdjecie = cv2.imread('moje_zdjecie.png')
szareZdjecie = cv.cvtColor(img, cv2.COLOR_BGR2GRAY) # Konwersja obrazu do postaci zdjęcia w odcieniach szarości

szareZdjecie = np.float32(szareZdjecie) # Konwersja obrazu do postaci zdjęcia w formacie float32
harris = cv2.cornerHarris(szareZdjecie,2,3,0.04)
```

(2)

```
# blockSize (np. 2) - Jest to wielkość sąsiedztwa brana pod uwagę przy wykrywaniu narożników
# ksize (np. 3) - Parametr wielkości maksi używanej pochodnej Sobela.
# k (np. 0.004) - Parametr wolny w równaniu detektora Harris (0.004 – 0.006)

zdjecie [harris >0.01* harris.max()]=[0,0,255]
 cv2.imshow('Punkty wykryte detektorem Harrisa', zdjecie)
 if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Czasami może być potrzebne znalezienie narożników z dokładnością podpikselową, wykorzystywana jest do tego cv2.cornerSubPix(). Na rysunku poniżej przedstawiono wyniki wykorzystania funkcji cv2.cornerSubPix



```
#Import biblioteki OpenCV I NumPy
import cv2
import numpy as np

zdjecie = cv2.imread('moje_zdjecie.png')
szareZdjecie = cv.cvtColor(img, cv2.COLOR_BGR2GRAY) # Konwersja obrazu do postaci zdjęcia w odcieniach szarości

szareZdjecie = np.float32(szareZdjecie) # Konwersja obrazu do postaci zdjęcia w formacie float32
harris = cv2.cornerHarris(szareZdjecie,2,3,0.04)

# blockSize (np. 2) - Jest to wielkość sąsiedztwa brana pod uwagę przy wykrywaniu narożników
# ksize (np. 3) - Parametr wielkości maksi używanej pochodnej Sobela.
# k (np. 0.004) - Parametr wolny w równaniu detektora Harris (0.004 – 0.006)

# find Harris corners
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)
harris = cv2.dilate(harris,None)
ret, dst = cv2.threshold(harris,0.01* harris.max(),255,0)
dst = np.uint8(dst)
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)

# define the criteria to stop and refine the corners
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.001)
corners = cv2.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)

res = np.hstack((centroids,corners))
res = np.int0(res)
zdjecie [res[:,1],res[:,0]]=[0,0,255]
zdjecie [res[:,3],res[:,2]] = [0,255,0]

cv2.imwrite('zdjecia.png', zdjecie)
```

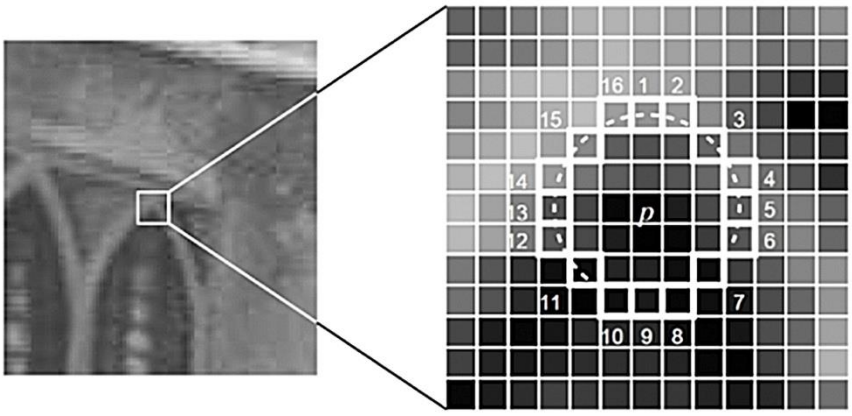
(3)

1.1.2. Detektor FAST

Koncepcja działania detektora FAST (ang. *Features from Accelerated Segment Test*), przedstawiona przez Rostena oraz Drummonda w 2006 roku, opiera się na założeniu, że punkty charakterystyczne posiadają jasno zdefiniowaną pozycję oraz są nośnikiem dobrze rozpoznawalnej informacji, pozwalającej na ich jednoznaczne wykrywanie na sąsiednich zdjęciach. Zaletą detektora FAST jest szybkość przetwarzania zdjęć, gdyż został zaprojektowany w celu wykrywania punktów wiążących w czasie rzeczywistym, np. w przypadku orientacji zdjęć z wykorzystaniem algorytmu SLAM.

Działanie algorytmu FAST składa się z pięciu głównych kroków:

1. wybranie na zdjęciu odpowiedniego piksela, który zostanie poddany analizie pod kątem przynależności do grupy punktów traktowanych jako narożnik. Wartość intensywności wyznaczonego punktu jest oznaczana symbolem I_p,
2. określenie akceptowalnej wartości progu, poniżej której punkty są odrzucane,
3. zdefiniowanie okręgu o promieniu 16 pikseli, dla którego będą analizowane wartości pikseli (Rys. 14),
4. zdefiniowanie początkowo wybranego punktu jako narożnika – jeżeli w jego sąsiedztwie (czyli okręgu o obwodzie 16 pikseli) znajduje się *n* pikseli o intensywności większej, bądź równej od I_p, to wybierane jest 12 z nich,
5. wyselekcjonowanie i porównanie pikseli oznaczonych numerami: 1, 3, 5 oraz 13, co ma na celu przyspieszenie działania pierwszego etapu algorytmu. W przypadku gdy dwa pierwsze piksele są ciemniejsze lub jaśniejsze od porównywanego piksela, to algorytm przechodzi do następnego kroku, czyli porównania kolejnych dwóch pikseli. Dla piksela stanowiącego potencjalny narożnik, niezbędne jest sprawdzenie, czy trzy piksele otaczające są jaśniejsze lub ciemniejsze. Jeżeli oba przedstawione warunki zostają spełnione, to porównywane są kolejne piksele w celu sprawdzenia poprawności przyjętej hipotezy.



Rysunek 3 Przykład detekcji narożnika wraz z okręgiem 16 pikseli poddanych dalszej analizie dla detektora FAST[4]

Duża szybkość działania algorytmu wiąże się z istotnym ograniczeniem – algorytm bywa niestabilny i nie zawsze działa poprawnie. Dla przykładu, jeżeli w oknie poszukiwawczym wykrytych zostało mniej niż 12 punktów, zbyt wiele punktów uznawanych jest za narożniki.

[4] Edward Rosten and Tom Drummond, 'Machine Learning for High Speed Corner Detection', *Computer Vision -- ECCV 2006*, 1 (2006), 430–43 <https://doi.org/10.1007/11744023_34>.

Co więcej, czas trwania procesu wyboru punktów inicjalnych determinuje całkowity czas wyszukiwania punktów – narożników.

```
#Import biblioteki OpenCV
import cv2
import numpy as np

zdjecie = cv2.imread('moje_zdjecie.png')
szareZdjecie = cv2.cvtColor(zdjecie, cv2.COLOR_BGR2GRAY) # Konwersja obrazu do postaci zdjęcia w odcieniach szarości

# Inicjalizacja detektora FAST z wartościami domyślnymi funkcji
fast = cv2.FastFeatureDetector.create()

# Wykrycie punktów charakterystycznych (keypoints)
kp = fast.detect(szareZdjecie, None)

# Narysowanie punktów charakterystycznych (keypoints)
img2 = cv2.drawKeypoints(szareZdjecie, kp, None, color=(255,0,0))

# Wyświetlenie parametrów funkcji
print("Threshold: ", fast.getThreshold())
print("nonmaxSuppression: ", fast.getNonmaxSuppression())
print("neighborhood: ", fast.getType())
print("Total Keypoints without nonmaxSuppression: ", len(kp))
```

(4)

Non-maximal Suppression

Jednym z problemów wykorzystania algorytmu FAST jest duża liczba wykrytych punktów charakterystycznych traktowanych jako narożniki.. W celu wyboru najlepszych punktów charakterystycznych (stabilnych według teorii Mikołajczyka i Lowe), wykorzystuje się podejście Non-maximum Suppression, które zakłada następujące kroki:

- Obliczana jest funkcję kosztu, V dla wszystkich wykrytych punktów charakterystycznych.. V jest sumą bezwzględnych różnic pomiędzy wartościami p i 16 otaczających go pikseli.
- Analizowane są dwa sąsiadujące punkty kluczowe , dla których wyznaczane są wartości V.
- Odrzucany jest punkt dla którego uzyskano niższą wartość funkcji kosztu V.

W kodzie nr. 5 zamieszczono przykład wykorzystania funkcji Non-maximum Suppression oraz zmiany pozostałych parametrów funkcji FAST:

```
#Import biblioteki OpenCV
import cv2
import numpy as np

zdjecie = cv2.imread('moje_zdjecie.png')
szareZdjecie = cv2.cvtColor(zdjecie, cv2.COLOR_BGR2GRAY) # Konwersja obrazu do postaci zdjęcia w odcieniach szarości

# Inicjalizacja detektora FAST z wartościami domyślnymi funkcji
fast = cv2.FastFeatureDetector.create(threshold = 10, nonmaxSuppression = True, type = cv2.FAST_FEATURE_DETECTOR_TYPE_9_16) # cv2.FAST_FEATURE_DETECTOR_TYPE_5_8, cv2.FAST_FEATURE_DETECTOR_TYPE_7_12

# Wykrycie punktów charakterystycznych (keypoints)
kp = fast.detect(szareZdjecie, None)
```

(5)

```
# Narysowanie punktów charakterystycznych (keypoints)
img2 = cv2.drawKeypoints(szareZdjecie, kp, None, color=(255,0,0))

# Wyświetlenie parametrów funkcji
print("Threshold: ", fast.getThreshold())
print("nonmaxSuppression: ", fast.getNonmaxSuppression())
print("neighborhood: ", fast.getType())
print("Total Keypoints without nonmaxSuppression: ", len(kp))
```

1.2. Detektory powierzchni BLOB

Detektory BLOB stanowią ważną grupę detektorów wykorzystywanych do wyszukiwania charakterystycznych obszarów w oparciu o określone właściwości otoczenia (np. jasność czy kolor). W tym wypadku badany obszar powinien się charakteryzować następującymi właściwościami:

- dużą kontrastowością w stosunku do najbliższego otoczenia,
- dobrą definiowalnością w przestrzeni zdjęcia,
- powtarzalnością niezależnie od warunków (inwariantnością) i zmian np. oświetlenia czy zniekształceń geometrycznych (skręcenia, skali itp.),
- odpornością na występowanie szumu.

Istotą orientacji danych z wykorzystaniem cech wykrytych na obrazie jest wykrycie stabilnych punktów kluczowych oraz ich opisanie za pomocą deskryptorów. Niezbędne jest, aby wykryte punkty pozbawione były szumów oraz były inwariantne na zmiany, czyli pozostawały wolne od wpływu takich czynników jak: skala, rotacja obrazu, oświetlenie czy modyfikacje położenia środków rzutów[5]. Najczęściej stosowanymi algorytmami przynależącymi do grupy BLOB są algorytmy SURF i SIFT[6]. Podobnie jak w przypadku wszystkich pozostałych algorytmów, także i one zostały zaprojektowane dla rastrów odwzorowanych w rzucie środkowym (obrazów), co z kolei wiąże się z występowaniem standardowych zniekształceń obrazu na kolejnych seriach zdjęć. W celu wykrycia na zdjęciach obszarów charakterystycznych wykorzystywane jest zwykle przekształcenie DoG (ang. *Difference of Gaussian*) (DoG; [7]), zaimplementowany w SIFT [8] lub DoH (ang. *Determinant of Hessian*), stosowany w SURF [9].

```
#Import biblioteki OpenCV I NumPy
import cv2
import numpy as np

zdjecie = cv2.imread('moje_zdjecie.png')
szareZdjecie = cv.cvtColor(zdjecie, cv2.COLOR_BGR2GRAY) # Konwersja obrazu do postaci zdjęcia w odcieniach szarości

# Inicjalizacja detektora  simple blob detector
detector = cv2.SimpleBlobDetector()
```

(6)

[5] S. Urban and M. Weinmann, 'Finding a Good Feature Detector-Descriptor Combination for the 2D Keypoint-Based Registration of Tls Point Clouds', *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W5 (2015), 121–28 <https://doi.org/10.5194/isprsannals-II-3-W5-121-2015>.

[6] Urban and Weinmann.

[7] Lindeberg, 1993

[8] D.G. Lowe, 'Object Recognition from Local Scale-Invariant Features', *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, 1150–57 vol.2 <https://doi.org/10.1109/ICCV.1999.790410>.

[9] (Bay, Tuytelaars i Van Gool, 2006)

```
# Wykrywanie blobów
punktyCharakterystyczne = detector.detect(szareZdjecie)

# Wyświetlanie punktów charakterystycznych w postaci okręgów
zdjecieZzaznaczonymiBlobami = cv2.drawKeypoints(zdjecie, punktyCharakterystyczne, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow("Zdjecie z zaznaczonymi blobami", zdjecieZzaznaczonymiBlobami)
cv2.waitKey(0)

# Zmiana parametrów detektora SimpleBlob detektor
parametry = cv2.SimpleBlobDetector_Params()

# Zmiana wartości progu binaryzacji
parametry.minThreshold = 10;
parametry.maxThreshold = 200;

# Filtracja po polu powierzchni.
parametry.filterByArea = True
parametry.minArea = 1500

# Filtracja względem odstępstwa od okręgu
parametry.filterByCircularity = True
parametry.minCircularity = 0.1

# Filtracja względem wypełnienia blobu
parametry.filterByConvexity = True
parametry.minConvexity = 0.87

# Filtracja po stosunku pół osi elipsy
parametry.filterByInertia = True
parametry.minInertiaRatio = 0.01

detector = cv2.SimpleBlobDetector_create(parametry)
```

1.2.1. Detektor SIFT

Detektor SIFT (Scale Invariant Feature Transform) był jednym z pierwszych algorytmów umożliwiających wykrywanie regionów tzw. BLOB-ów, charakteryzujących się niezmiennymi cechami bez względu na skale obrazów oraz wpływ oświetlenia Głównym założeniem algorytmu SIFT jest zastosowanie tzw. detektora Difference – of – Gaussian (DoG; Lindberg, 1993). Jego głównym założeniem jest to, że potencjalne punkty dla cech lokalnych (niezależne od skali i orientacji) odpowiadają lokalnym ekstremom na obrazach różnicowych uzyskanych po filtracji filtrem Gaussa w różnych skalach. Działanie algorytmu SIFT składa się z następujących etapów:

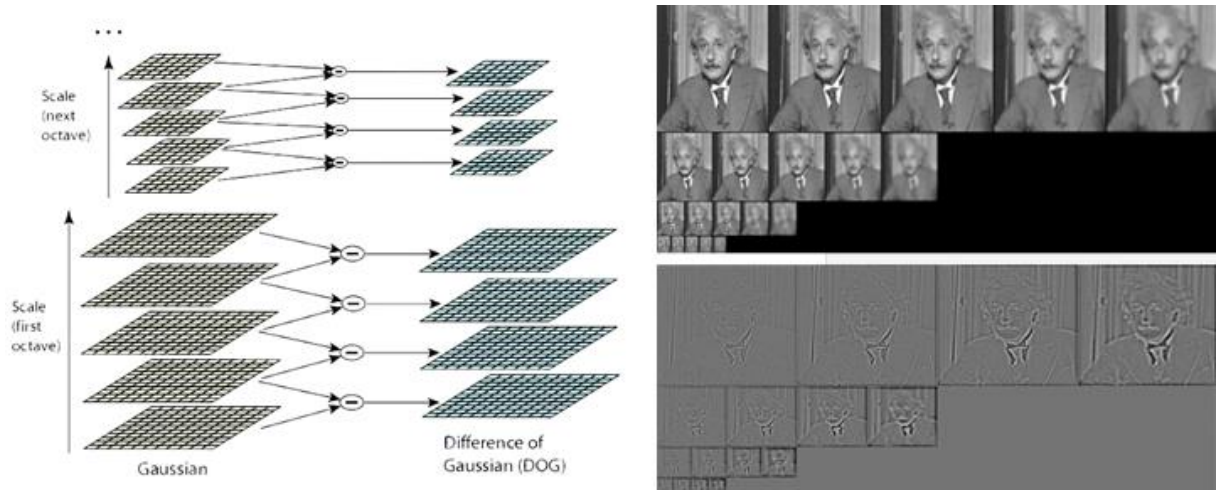
Wykrywanie ekstremów funkcji w różnych skalach

W pierwszym etapie wykrywania ekstremów funkcji w przestrzeni skalowania odbywa się poprzez porównanie obrazu oryginalnego z sąsiadującymi obrazami utworzonymi poprzez zastosowanie przekształcenia Gaussa. Przyjmuje się, że potencjalne punkty dla cech lokalnych (niezależne od skali i orientacji) odpowiadają lokalnym ekstremom na obrazach różnicowych uzyskanych po filtracji filtrem Gaussa w różnych skalach. Oryginalny obraz

I(x,y) jest poddawany filtracji przy wykorzystaniu funkcji Gaussa przedstawionej za pomocą wzoru:

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

Obraz wejściowy filtrowany jest filtrem Gaussa w różnych skalach rozdzielonych stałym współczynnikiem k, a następnie obliczane są obrazy różnicowe DoG (Different of Gaussian) Tę procedurę powtarza się przepróbkowując obraz poprzez wyrzucenie co 2 wiersza i kolumny.



Rysunek 4 A) Przykład działania filtracji funkcja DoG (Difference of Gaussian); B) Przykład obrazu przetworzonego filtrem Gaussowskim i DoG na różnych poziomach.

Funkcję DoG (Difference of Gaussian) można przedstawić za pomocą wzoru:

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

Wykrywanie lokalnych ekstremów funkcji odbywa się poprzez porównanie punktu z jego sąsiadami w otoczeniu 3x3x3. Punkt jest kwalifikowany jako kandydat na punkt kluczowy cechy, gdy ma wartość większą lub mniejszą od punktów sąsiednich.

Lokalizacja punktów kluczowych (ang. keypoints)

Kolejnym ważnym krokiem związanym z dzianiem detektora jest wykrycie tzw. punktów kluczowych (keypoints). Lokalizacja punktów kluczowych odbywa się poprzez porównanie wartości funkcji dopasowania, obliczonej na podstawie ekstremów:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x, x = (x, y, \sigma)^T$$

gdzie D i pochodne tej funkcji obliczane są dla danego punktu, a x jest przesunięciem względem tego punktu. Położenie ekstremum wylicza z funkcji:

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

W celu eliminacji punktów o słabym kontraście wykorzystuje się poniższą funkcję. Przyjmuje się, że gdy wartość ekstremum jest mniejsze niż 0.03 (w przypadku wartości intensywności
z przedziału {0,1}), to dany punkt posiada słaby kontrast (Lowe, 2004).

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Dzięki wykorzystaniu powyższej zależności możliwe jest usunięcie punktów leżących na linii. Wykorzystując funkcję DoG, możemy się spodziewać, że algorytm będzie mieć silną odpowiedź wzdłuż krawędzi, nawet jeżeli krawędź jest słabo odwzorowana na obrazie. W celu wyznaczenia głównych krzywizn należy wykorzystać Hessian:

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

Pod pojęciem Hessianu rozumiemy macierz kwadratową drugich pochodnych cząstkowych funkcji o wartościach rzeczywistych dwukrotnie różniczkowalnej w pewnym punkcie dziedziny.

Wartości własne H są proporcjonalne do głównych krzywizn D. Wykorzystując podejście zaproponowane przez Harrisa i Stephensa (1988), możemy wykorzystać stosunek uzyskanych wartości własnych macierzy H. Przyjmując α jako największą wartość własną macierzy, a β jako najmniejszy możemy wyznaczyć wartości śladu macierzy i jej wyznacznika:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$
$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta 2)$$

W przypadku, gdy wyznacznik ma wartość ujemną, krzywizny mają różne znaki, a więc punkt jest odrzucany, ponieważ nie jest on maksimum funkcji. Przyjmując r jako stosunek największej do najmniejszej wartości własnej macierzy możemy zapisać to w postaci zależności: $\alpha = r\beta$, stąd:

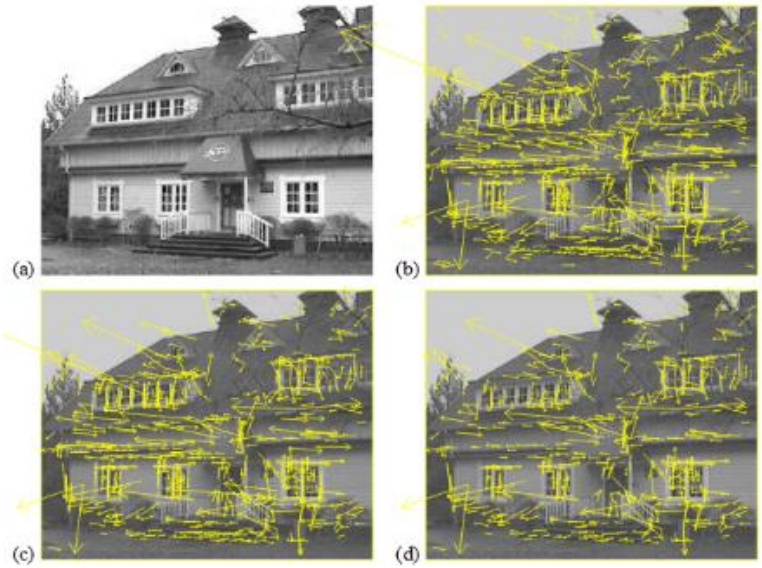
$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

Ta zależność przyjmuje wartości minimalne w przypadku, gdy dwie wartości własne macierzy są sobie równe, natomiast wzrasta gdy wzrasta wartość r. W celu sprawdzenia, czy wyszukane punkty nie leżą na linii, należy jedynie sprawdzić, czy zostaje spełniony warunek przy założonej wartości progu r:

$$\frac{Tr(H)^2}{Det(H)} \leq \frac{(r+1)^2}{r}$$

Zastosowanie tego rozwiązania jest bardzo efektywne ze względu na fakt, iż niezbędne jest wykonanie mniej niż 20 operacji w celu sprawdzenia poprawności doboru punktu charakterystycznego. Lowe zaproponował w swojej publikacji zarekomendował używanie wartości r = 10 (Lowe, 2004).

Na rysunku 16 przedstawione zostały wyniki działania algorytmu detekcji punktów przy wykorzystaniu współczynnika filtracji $D(\hat{x})$ = 0.03 oraz $\frac{(r+1)^2}{r}$ = 10. W wyniku tego procesu z początkowo wykrytych 832 punktów, odfiltrowano 102 punkty charakteryzujące się niskim kontrastem i prawdopodobnie znajdujące się na liniach.



Rysunek 5 Etapy wyboru punktów charakterystycznych a) Oryginalne zdjęcie w rozdzielczości 233x189 pikseli, b) Wstępnie wykrytych 832 punktów charakterystycznych (keypoints), bazujących na maximach i minimach wartości funkcji Difference of Gaussian. Wykryte punkty wiążące zaznaczono jako wektory ze skalą i orientacją c) Wyniki filtracji punktów charakterystycznych (729) przy zastosowaniu progu filtracji 0.003, d) Wyniki filtracji punktów charakterystycznych (536) leżących na linii ((Lowe, 2004).).

```
#Import biblioteki OpenCV I NumPy
import cv2
import numpy as np

zdjecie = cv2.imread('moje_zdjecie.png')
szareZdjecie = cv2.cvtColor(zdjecie, cv2.COLOR_BGR2GRAY) # Konwersja obrazu do postaci zdjęcia w odcieniach szarości

sift = cv2.SIFT_create()
kp = sift.detect(gray,None)
zdjecie =cv2.drawKeypoints(zdjecie , kp,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imwrite('sift_keypoints.jpg', zdjecie)
# Zmiana parametrów detektor SIFT

sift = cv2.SIFT_create(nfeatures = 0, nOctaveLayers = 3, contrastThreshold = 0.04, edgeThreshold = 10, sigma = 1.6)

#nfeatures -  Liczba najlepszych cech do zachowania. Cechy są uszeregowane według ich punktacji (mierzonej w algorytmie SIFT jako lokalny kontrast)
```

(7)

#nOctaveLayers Liczba warstw w każdej oktawie. 3 to wartość użyta w pracy D. Lowe. Liczba oktaw jest obliczana automatycznie na podstawie rozdzielczości obrazu.

#contrastThreshold Próg kontrastu używany do odfiltrowania słabych cech w regionach o niskim kontraście. Im większy próg, tym mniej cech jest produkowanych przez detektor. Próg kontrastu zostanie podzielony przez nOctaveLayers, gdy zastosowane zostanie filtrowanie. Gdy nOctaveLayers jest ustawione jako domyślne i jeśli chcesz użyć wartości użytej w pracy D. Lowe, 0.03, ustaw ten argument na 0.09.

#edgeThreshold Próg używany do odfiltrowania cech leżących na krawędziach -. im większy edgeThreshold, tym mniej cech jest odfiltrowywanych (więcej cech jest zachowywanych).

#sigma Sigma Gaussian zastosowany do obrazu wejściowego w oktawie #0.

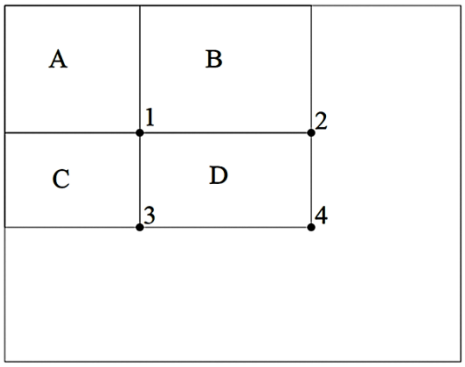
1.2.2. Detektor SURF

W 2006 został zaprezentowany przez Bay'a *i in* detektor SURF (ang. *Speeded Up Robust Features*), będący modyfikacją algorytmu SIFT. Autorzy założyli, że działanie nowego algorytmu będzie szybsze niż algorytmu SIFT i oparli go na wykorzystaniu macierzy Hesjanu. Działanie SURF składa się z czterech etapów, z których pierwsze dwa dotyczą detekcji punktów charakterystycznych, a dwa kolejne odnoszą się do opisywania punktów przy wykorzystaniu deskryptora.

Główne założenia algorytmu SURF obejmują:

Obliczanie obrazu typu *integral* [10]

Transformacja obrazu z postaci pikselowej do postaci cech (ang. *features*) poprzez obliczenie obrazów *integral*, umożliwia przyspieszenie prac związanych z wykrywaniem punktów charakterystycznych na zdjęciach.



Rysunek 6 Przykład przekształcenia obrazu w obraz „I". Wartość zdjęcia integral w punkcie 1 jest sumą pikseli w kwadracie A. Wartość w punkcie 2 jest równa sumie wartości w kwadratach A i B, w punkcie 3 równa się sumie A i C, a w punkcie 4 wynosi A+B+C+D [11]

Obraz *integral* w punkcie *(x,y)* powstaje w wyniku zsumowania punktów znajdujących się w rzędzie powyżej oraz z lewej strony punktu o współrzędnych *x,y* (**Błąd! Nie można odnaleźć źródła odwołania.**). Zmianę obrazu z postaci *pikselowej* do postaci *integral* można wykonać przy wykorzystaniu wzoru:

[10] P. Viola and M. Jones, 'Rapid Object Detection Using a Boosted Cascade of Simple Features', *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 1 (2001), I-511-I-518 <https://doi.org/10.1109/CVPR.2001.990517>.

[11] Herbert Bay and Andreas Ess, 'Speeded-Up Robust Features (SURF)', 110.September (2008), 346–59 <https://doi.org/10.1016/j.cviu.2007.09.014>.

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y')$$

gdzie: *ii(x,y)* jest obrazem *integral*, a *i(x,y)* jest obrazem oryginalnym. Wykorzystując wzory:

$$s(x,y) = s(x,y-1) + i(x,y)$$
$$ii(x,y) = ii(x,y-1) + s(x,y)$$

gdzie *s(x,y)* jest sumą wartości w jednym rzędzie, możliwe jest obliczenie obrazu *integral* w jednym kroku dla całego zdjęcia.

Lokalizacja punktów kluczowych

Ważnym krokiem związanym z działaniem detektora jest wykrycie tzw. punktów kluczowych. Lokalizacja punktów odbywa się przy wykorzystaniu Hesjanu macierzy, który znacząco wpływa na przyspieszenie czasu wyszukiwania punktów wiążących oraz poprawę dokładności ich detekcji. Wykryty punkt charakterystyczny odpowiada maksimum wyznacznika macierzy *H(x,y,σ)*:

$$H(x,y,\sigma) = \begin{bmatrix} L_{xx}(x,y\sigma) & L_{xy}(x,y\sigma) \\ L_{xy}(x,y\sigma) & L_{yy}(x,y\sigma) \end{bmatrix}$$

gdzie: x,y jest położeniem punktu w skali σ oraz $L_{xx}(x,y\sigma)$ – obrazem splotu pochodnej cząstkowej drugiego stopnia funkcji Gaussa $D_{xx} = \frac{\partial^2}{\partial x^2}\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ i obrazu I w punkcie x,y.

Do wyznaczenia punktów charakterystycznych służy okno poszukiwawcze o rozmiarze 9x9 pikseli, w którym brane są pod uwagę wartości *σ* dla najniższego poziomu piramidy. Pod pojęciem piramidy obrazu rozumie się zapis zdjęcia, które zostało przetworzone filtrem Gaussa z usunięciem co n–tego piksela w zależności od przyjętego stopnia szczegółowości i rozdzielczości badanego obrazu. Takie podejście stosowane jest w przypadku przetwarzania zdjęć detektorem SIFT. Przetwarzanie zdjęć detektorem SURF wiąże się z wykorzystaniem filtrów prostokątnych oraz obrazów *integral*, których główną zaletą jest brak konieczności iteracyjnego zastosowania tego samego filtra na wielu zdjęciach. Dzięki temu możliwe jest zastosowanie różnych filtrów na różnych poziomach piramidy obrazów, a zatem proces skalowania wielkości filtra (prostokątnego) może zastąpić skalowanie obrazu poprzez usuwanie określonych pikseli. W pierwszym etapie przyjmuje się wartość maski 9x9 oraz wartość skali *s=1,2*, które odpowiadają parametrowi filtracji Gaussowskiej *σ = 1,2*. Następnie zwiększana jest wielkość maski, kolejno do rozmiarów 9x9, 15x15, 21x21, 27x27 itd. Przy większych skalach maska powinna być powiększana o stałą dodawania, dwukrotnie zwiększaną (*n*x6), podobnie jak zmianom ulega wartość współczynnika *σ* – przykładowo dla filtru 27x27 wartość współczynnika σ będzie zwiększona 3x1,2 = 3,6.

W celu wykrycia punktów charakterystycznych na obrazie i obrazach pochodnych w innych skalach analizowane są obszary 3x3x3, a następnie wyznaczane są maksima wyznacznika Hesjanu, które są interpolowane w różnych skalach w oparciu o metodę Browna i Lowe'a [12].

[12] Matthew Brown and David G Lowe, 'Invariant Features from Interest Point Groups', *In British Machine Vision Conference*, 2002, 656–65 <https://doi.org/10.1.1.1.8475>.

1.2.3. Detektor MSER

Innym podejściem do wyszukiwania cech charakterystycznych na zdjęciach jest podejście bazujące na wykrywaniu obszarów charakterystycznych. Detektor MSER (Maximally Stable Extremal Regions) został przedstawiony przez Matas'a, Chum'a, Urban'a i Pajdla w 2002 roku. Pod pojęciem regionu rozumiany jest mały obszar na zdjęciu oraz jego najbliższe otoczenie jednoznacznie rozpoznawalne na zdjęciu. W celu jednoznacznego zdefiniowania przydatności badanego regionu jako regionu charakterystycznego spełnione muszą zostać następujące warunki:

- powinien wyróżniać się w „lokalnym rejonie" (kontrast z najbliższym otoczeniem)
- powinien być niezależny od zmiany wartości intensywności na całym zdjęciu,
- powinien być stabilny przy „wirtualnym" przetwarzaniu,
- powinien być powtarzalny przy zastosowaniu różnych skal obrazu
- powinien być obliczany w przestrzeni O(n log log$n$), gdzie n jest liczba pikseli

Koncepcja MSER zakładająca, że każde zdjęcie I da się zamienić do postaci binarnej E_t, przy wykorzystaniu pewnych współczynników thresholding:

$$E_t(x) = \begin{cases} 1\ jeśli\ I(x) \geq t \\ 0\ w\ przeciwnym\ przypadku \end{cases}$$

Przyjmuje się, że piksele powyżej wartości filtrującej będą przyjmować kolor czarny, natomiast
w przypadku wartości mniejszej kolor biały. Wpływ zmiany współczynnika threshold można przedstawić na podstawi koncepcji filmu. Każda klatka filmu przedstawia inny obraz powstały w wyniku binaryzacji badanego obrazu. W początkowej fazie zastosowanego przetwarzania zdjęcia uzyskiwany jest biały obraz. Wraz ze zmianą wartości współczynnika t pojawiać się będą coraz to nowe kropki, aż do momentu kiedy nie powstanie cały czarny obraz. Podobny proces powtarza się dla zdjęcia w odwróconych brawach. Nałożenie maksimów i minimów z dwóch obrazów umożliwia wykrycie charakterystycznych regionów na zdjęciach. W przypadku zastosowania pozytywu wyszukiwane są ciemne regiony nazywane MSER+, a przy zastosowaniu negatywu wyszukiwane są obszary jasne MSER- (Rys 18).

Rysunek 7 Przykład wykrytych regionów zastosowaniem algorytmu MSER. Białe elipsy przedstawiają wykryte jasne regiony (MSER-), czarne elipsy przedstawiają ciemne wykryte regiony (MSER+)

Przyjmuje się, że jeżeli algorytm wykryje i uzna za stabilne dla minimum 7 różnych thresholdów, to dany obszar może być uznawany jako obszar charakterystyczny. Taki obszar nazywany jest *margin of the region*.

```
mser = cv2.MSER_create()
regions = mser.detectRegions(img, None)
hulls = [cv2.convexHull(p.reshape(-1, 1, 2)) for p in regions]
cv2.polylines(vis, hulls, 1, (0, 255, 0))
cv2.imshow('img', vis)
cv2.waitKey(0)
cv2.destroyAllWindows()
```