Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Podstawy sztucznej inteligencji i uczenia głębokiego

Inżynieria Mechatroniczna

Instrukcja 8:

Cechy obrazu
i
Klasyfikacja obiektów

Dowiesz się: czym jest miara podobieństwa i jak znaleźć znany wzór na obrazie, czym jest współczynnik korelacji krzyżowej normalizacji, czym są obiekty w obrazie binarnym, czym jest etykietowanie, jak znaleźć podstawowe cechy obiektów za pomocą funkcji regionprops MATLABa, czym są cechy geometryczne: niezmienniki momentowe momenty Hu, jak zbudować wektor cech, jak ocenić wrażliwość cech na skalowanie i obrót, jak zbudować prosty klasyfikator obiektów, czym są cechy lokalne i ich zastosowania, jak zaimplementować i przetestować detektor naroży Harrisa

Materiały dodatkowe:

- Wykład kursowy
- Baza obrazów dostarczona przez wykładowcę

Opiekun przedmiotu:
Krzysztof Holak, holak@agh.edu.pl

Autor instrukcji:
Krzysztof Holak, holak@agh.edu.pl

Współczynnik korelacji obrazu do znajdowania obiektów

Współczynnik korelacji obrazu może być użyty do znalezienia danego wzorca. Funkcja korelacji w programie Matlab przyjmuje wzorzec w formie mniejszego obrazu i szuka go na wejściowym obrazie lub serii obrazów. Zmienna utworzona przez funkcję korelacji jest macierzą, nowym obrazem, który ilustruje stopień podobieństwa między wzorem a obszarem na obrazie umieszczonym pod wzorcem dla każdej pozycji wzorca na obrazie. Następnie, poprzez wyszukiwanie maksymalnej wartości w tej macierzy, można znaleźć lokalizację wzorca na obrazie. Aby obliczyć mapę podobieństwa między wzorcem a obrazem, należy użyć funkcji

```
c = normxcorr2(image_pattern,image);
```

Zmienną c można wyświetlić jako obraz (funkcje **image()** lub **imagesc()**) lub jako wykres powierzchniowy za pomocą polecenia **surf(c)**. Następnie, biorąc wartość podobieństwa jako funkcję położenia na obrazie, można znaleźć maksymalną wartość funkcji i lokalizację tego maksimum. W MATLAB-ie można to zrobić za pomocą kodu:

```
[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax(1));
```

Pamiętaj, że funkcja **normxcorr2()** zmienia rozmiar analizowanego obrazu. Dlatego, aby wyświetlić środek znalezionego wzorca, przeliczamy indeksy w następujący sposób:

```
peak_offset = [(xpeak - size (image_pattern,2))
               (ypeak - size (image_pattern,1))];
```

Po użyciu **surf()** należy zastosować polecenie **shading flat**, aby wyświetlić odpowiedź funkcji korelacji w postaci powierzchni, ponieważ w przeciwnym razie na rysunku zostanie narysowanych zbyt wiele krawędzi i cały wykres będzie czarny.

Oto prosty kod, który wyszukuje wzór na obrazku

```
clc;
clear all;
close all;
% Correlation for pattern detection examples
im = imread('Wagtail.JPG');
pattern = imread('Wagtail_P2.JPG');

imGray = rgb2gray(im);
patternGray = rgb2gray(pattern);
imGrayTemp = imGray;

figure,
subplot(1,2,1)
imshow(imGray);
subplot(1,2,2)
imshow(patternGray);

% image correlation computation - find one pattern in the image
c = normxcorr2(patternGray,imGray);

[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax(1));

peak_offset = [(xpeak - size (patternGray,2))
(ypeak - size (patternGray,1))];
```

```
figure,
subplot(1,2,1)
imagesc(c);
subplot(1,2,2)
surf(c);
shading flat

figure,
imshow(im); hold on
% plot(peak_offset(2),peak_offset(2),'*'); hold on
rectangle('Position',[peak_offset',size(patternGray)],'Edgecolor','red');
```

Korelacja obrazu stała się ważnym narzędziem w bezkontaktowych pomiarach wielkości mechanicznych, takich jak przemieszczenia, ugięcia i odkształcenia. Jest podstawą techniki pomiarowej zwanej DIC (Digital Image Correlation). Przeczytaj o tym w Internecie i spróbuj znaleźć rozwiązania dostępne na rynku. Jeśli ten temat Cię interesuje, możesz przeczytać więcej, dowiedzieć się więcej o algorytmie i pobrać i przetestować bezpłatną wersję programu opracowanego w środowisku obliczeniowym MATLAB na stronie http://www.ncorr.com/

Zadanie 3.1. Uruchom kod MATLABa, aby znaleźć wzór na obrazie. Zaznacz znaleziony wzór, otaczając go czerwonym prostokątem lub kropką reprezentującą jego środek. Zobacz, jak funkcja znajduje różne wzory. Obserwuj odpowiedź korelacji i omów, które ze wzorów są najlepsze do wykrywania i dlaczego. Który z nich byłby bardziej odporny na szum na obrazie?

Analiza obrazu – wykrywanie i klasyfikacja obiektów

Przypomnijmy, że pierwszym krokiem analizy obiektu jest binaryzacja. Poznałeś zarówno podstawowe, jak i bardziej zaawansowane metody. Po binaryzacji obiekty na obrazie muszą składać się z białych pikseli (o wartości 1), podczas gdy tło staje się czarne (piksele o wartości 0). Następnie wszystkie obiekty muszą zostać oznaczone za pomocą **`bwlabel()`**. Ta funkcja po prostu nadaje numery (etykiety) wszystkim oddzielnym obiektom na obrazie. W następnym kroku parametry geometryczne opisujące kształty są obliczane za pomocą funkcji **`regionprops()`**. W tym ćwiczeniu użyj tej funkcji, aby uzyskać wszystkie dostępne cechy – użyj właściwości ' all ' .

Oto prosty program z poprzednich zajęć, który oblicza wszystkie cechy obiektów.

```
clc;
clear all;
close all;

imRGB = imread('Objects.jpeg');

figure(1)
subplot(1,2,1)
imshow(imRGB);
title('Original Image')

imGray = rgb2gray(imRGB);

thresh = graythresh(imGray);
imBW = im2bw(imGray,thresh);

figure(1)
subplot(1,2,2)
imshow(imBW);
title('Binary Image')
```

```
imLB = bwlabel(imBW,4);
features = regionprops(imLB,'all');
```

Zmodyfikujmy program w następujących ćwiczeniach:

Zadanie 3.1 Zmień program w taki sposób, aby obliczane były wszystkie cechy, a nie tylko podstawowe. Użyj obrazu, w którym występuje kilka różnych kształtów. Wykonaj binaryzację tego obrazu – uzyskując obraz binarny ze wszystkimi obiektami. Napisz kod, który rysuje środek ciężkości każdego obiektu na obrazie i jego numer obok niego. Znajdź, jak różne cechy zmieniają się wraz z kształtem obiektu.

Cechy, które należy zbadać w tym ćwiczeniu: *Area, BoundingBox, Centroid, ConvexArea, Circularity, Eccentricity, EquivDiameter, EulerNumber, Extent, Orientation, MaxFeretProperties, MinFeretProperties, MajorAxisLength, MinorAxisLength, Solidity.*

Zadanie 3.2 Spróbuj znaleźć minimalny zestaw cech, który jest niezbędny do sklasyfikowania wszystkich kształtów na obrazie. Cechy nie powinny być redundantne, np. powinieneś odrzucić cechę, która niesie te same informacje co inne. Następnie napisz prosty kod do klasyfikacji kształtów na podstawie wartości cech, używając zagnieżdżonej struktury if-else.

Momenty i niezmienniki momentów

W przetwarzaniu obrazów binarnych momenty i niezmienniki momentów są pojęciami matematycznymi opisującymi cechy geometryczne obiektów, np. ich kształt. Są one ważne w dziedzinie rozpoznawania i klasyfikacji obiektów oraz analizy kształtu.

Momenty – Momenty to liczby związane z obiektami, które określają przestrzenny rozkład ich pikseli i mogą być używane do opisu kształtu. Istnieje kilka typów momentów, które zostaną opisane w poniższej instrukcji laboratoryjnej.

1. **Zwykłe momenty** – opisują ogólne właściwości geometryczne (rozkład przestrzenny pikseli) obiektów na podstawie ich kształtu (rozkład pikseli) w wybranych układzie współrzędnych.

$$M_{pq} = \sum_x \sum_y x^p y^q I(x,y)$$

   Gdzie I(x,y) – to jasność pikseli w punkcie (x, y) – w przypadku obrazów binarnych przyjmuje ona wartość 1 dla pikseli obiektu, a p, q – to liczby całkowite s oznaczające rząd momentów.

2. **Momenty centralne** – opisują przestrzenny rozkład pikseli obiektu względem jego środka ciężkości. Są **odporne** na zmienę **położenia** obiektu na obrazie. Aby obliczyć momenty centralne, najpierw należy obliczyć położenie środka ciężkości obiektu, używając zwykłych momentów pierwszego rzędu. Następnie momenty centralne oblicza się w następujący sposób

$$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q I(x,y)$$

Wzór jest taki sam jak poprzednio, z tą różnicą, że należy odjąć współrzędną centroidu x i y, jak pokazano w równaniu.

3. **Znormalizowane momenty** – te momenty są **odporne** na **zmiany skali** obiektów. Są obliczane na podstawie momentów centralnych i powierzchni obiektów. Są to momenty centralne podzielone przez współczynnik skalowania.

$$\varphi_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{1+\frac{p+q}{2}}}$$

4. **Niezmienniki momentowe** – są to rodzaje momentów, które są niezmienne pod wpływem przekształceń obrazu, takich jak translacja, skalowanie i obrót. Pomagają one rozpoznać ten sam typ obiektu na obrazie, nawet jeśli jest on zmieniony przez te przekształcenia.

   Jednym z najczęściej używanych niezmienników momentowych są **momenty Hu**. Zostały one opisane w literaturze naukowej przez MK Hu w 1962 r. Dla każdego obiektu obliczany jest zestaw 7 momentów Hu na podstawie znormalizowanych momentów centralnych. Momenty Hu są niezmienne wobec transformacji translacji, rotacji i skalowania.

W poniższym zestawie ćwiczeń zastosujesz funkcje MATLAB-a udostępnione przez instruktora w celu rozpoznania i klasyfikacji obiektów.

Oto kod funkcji obliczającej momenty i niezmienniki momentów

Najpierw po określeniu progu i etykietowaniu obrazu należy użyć funkcji **`regionprops()`**, aby obliczyć środki ciężkości i uzyskać listę pikseli należących do każdego z obiektów.

```
cechy = regionprops(imLB, 'Centroid', 'PixelList');
```

Następnie możesz obliczyć momenty, na przykład w przypadku momentów centralnych:

```
order = [3 3];

for k = 1:length(features)

    % find centroids and pixel list for each of the objects
    centroid(k).coordinates = features(k).Centroid;
    pixelList(k).List = features(k).PixelList;

    % Compute central moments for the object
    moments(k).CentralMoments = computeCentralMoments(pixelList(k).List,
centroid(k).coordinates,order);

end
```

Funkcje, które obliczają wszystkie momenty i niezmienniki momentów

```
function moments = computeCentralMoments(pixelList, centroid,order)

moments = zeros(order(1)+1, order(2)+1);  % Initialize a matrix for central moments
(p=0..2, q=0..2)
    c_x = centroid(1);
    c_y = centroid(2);

    % Calculate central moments for p, q = 0, 1, 2, 3, etc.

for i = 1:size(pixelList, 1)
        x = pixelList(i, 1);
        y = pixelList(i, 2);

        for p = 0:order(1)
            for q = 0:order(2)
                moments(p+1, q+1) = moments(p+1, q+1) + (x - c_x)^p * (y - c_y)^q ;
            end
        end
    end
end
```

Aby obliczyć momenty Hu, należy najpierw znormalizować momenty centralne uzyskane w poprzednim kroku.

```
function normalizedMomentsList = computeNormalizedMoments(momentsList)

for k = 1:length(momentsList)

    moments = momentsList(k).CentralMoments;
    normalized_moments = zeros(size(moments,1), size(moments,2));

    m00 = moments(1, 1);

    for p = 0:size(moments,1)-1
        for q = 0:size(moments,2)-1

            normalized_moments(p+1, q+1) = moments(p+1, q+1) / (m00^(1 + (p + q)/2));

        end
    end

    normalizedMomentsList(k).NormalizedMoments = normalized_moments;
end

end
```

Teraz możesz obliczyć zestaw siedmiu momentów Hu:

```
function hu_moments = computeHuMoments(normalizedMomentsList)

for k = 1:length(normalizedMomentsList)

normalized_moments = normalizedMomentsList(k).NormalizedMoments;

    eta20 = normalized_moments(3, 1);
    eta02 = normalized_moments(1, 3);
    eta11 = normalized_moments(2, 2);
    eta30 = normalized_moments(4, 1);
    eta03 = normalized_moments(1, 4);
    eta21 = normalized_moments(3, 2);
    eta12 = normalized_moments(2, 3);

    % Compute the 7 Hu moments
    phi1 = eta20 + eta02;
```

```
    phi2 = (eta20 - eta02)^2 + 4 * eta11^2;
    phi3 = (eta30 - 3 * eta12)^2 + (3 * eta21 - eta03)^2;
    phi4 = (eta30 + eta12)^2 + (eta21 + eta03)^2;
    phi5 = (eta30 - 3 * eta12) * (eta30 + eta12) * ((eta30 + eta12)^2 - 3 * (eta21 +
eta03)^2) + (3*eta12 - eta03)*(eta12 + eta03)*(3*(eta30 + eta12)^2 - (eta21 +
eta30)^2);
    phi6 = (eta20 - eta02) * ((eta30 + eta12)^2 - (eta21 + eta03)^2) + 4 * eta11 *
(eta30 + eta12) * (eta21 + eta03);
    phi7 = (3 * eta21 - eta03) *(eta30 + eta12)* ((eta30 + eta12)^2  - 3 *(eta12 -
eta30))-(eta30-3*eta12)*(eta21+eta03)*(3*(eta30 + eta12)^2 - (eta21 + eta30)^2);

    % Store Hu moments in a vector
    hu_moments(k).Hu = [phi1, phi2, phi3, phi4, phi5, phi6, phi7];

end

end
```

Zadanie 3.3 Przeanalizuj obrazy z tymi samymi obiektami geometrycznymi, ale z wprowadzonymi zmianami – translacją, obrotem i skalowaniem. Oblicz momenty centralne i znormalizowane. Oblicz momenty różnego rzędu dla każdego z obiektów i każdego z obrazów. Omów, jak wartość momentów zmienia się wraz z kształtami i jak zmienia się wraz z wprowadzonymi transformacjami. Które z momentów są niezmienne wobec translacji, skalowania i obrotu?

Zadanie 3.4 Powtórz ćwiczenie 3.3, uwzględniając momenty Hu w wektorze cech. Sprawdź niezmienność momentów Hu dla transformacji. Napisz program klasyfikacyjny, który wykorzystuje wszystkie momenty.

Zadanie 3.5 Zastosuj wiedzę zdobytą w poprzednim ćwiczeniu, aby napisać program do klasyfikacji prostych obiektów na podstawie wartości momentów. Program powinien klasyfikować wszystkie kształty obecne na obrazie. Narysuj centroidy każdego obiektu i zaznacz każdą klasę za pomocą pola ograniczającego o odrębnym kolorze.

Cechy lokalne i detektory cech

W wielu praktycznych przypadkach segmentacja obrazu oparta na regionach nie jest wystarczająca, aby właściwie opisać obiekty obecne w obrazie. Nawet najlepsze algorytmy mogą nie segmentować obrazu na obiekty z pożądanym poziomem dokładności. W takich przypadkach zamiast opisywać obiekty jako regiony, możemy traktować je jako zbiór punktów. Punkty mogą reprezentować charakterystyczne części obiektów, takie jak ich narożniki. Wykrywając te punkty i obliczając ich położenie, możemy modelować cały obiekt. Na przykład możemy użyć zbioru punktów do zidentyfikowania tego samego obiektu na serii obrazów, analizując relacje przestrzenne między punktami. Jest to możliwe nawet wtedy, gdy obiekt odkształca się podczas ruchu na wideo, ponieważ każdy z punktów może być śledzony indywidualnie. W literaturze poświęconej wizji komputerowej te charakterystyczne punkty są często nazywane **lokalnymi cechami obrazu** (cechami punktowymi).

W pakietach programistycznych dostępnych jest wiele różnych detektorów i deskryptorów cech. Różnią się one typami cech, które najlepiej wykrywają (**cechy typu narożninki vs cechy typu blob**), czasem i złożonością obliczeń oraz typem **niezmienności** (np. niezmienniczość rotacji i skali).

Oto kilka detektorów zaimplementowanych w MATLAB-ie:

1. Detektor Harrisa (1988) – najbardziej znany i szeroko stosowany w praktyce detektor narożników, wykrywający punkty narożne,
2. Algorytm minimalnej wartości własnej (algorytm Shi-Tomasiego, 1994) – detektor narożników wykorzystujący to samo podejście do wartości własnych co Harris, ale inną funkcję punktacji,
3. FAST (Features From Accelerated Segment Test, 2006) – detektor narożników, jest obliczeniowo szybki w porównaniu do innych detektorów, które wykorzystują podejście DoG (np. SIFT),
4. SURF (Speed-Up Robust Features, 2006) – wieloskalowy deskryptor cech, szybszy od bardziej znanego deskryptora SIFT, stosowany do wykrywania cech w kształcie plam w różnych skalach, przestrzeń skalowania generowana jest przy użyciu piramid obrazów,
5. ORB (Oriented FAST and Rotated BRIEF, 2011) – detektor, który jest kombinacją detektora narożników FAST w celu lokalizacji cechy i deskryptora BRIEF, ale jest niezmienny względem obrotu, jest odpowiedzią na opatentowany SIFT
6. BRISK (Binary Invariant Robust Scalable Keypoints, 2011) – kolejny deskryptor cech niezmiennych w skali, który jest odpowiedzią na opatentowany SIFT, jest to generalnie wieloskalowy detektor narożników FAST, nie jest niezmienny względem obrotów,
7. KAZE (2012) – deskryptor cech wieloskalowych, wykorzystuje nieliniowe filtrowanie dyfuzyjne do tworzenia przestrzeni skal (zamiast podejścia piramidalnego i rozmycia gaussowskiego), ma również otwarty kod źródłowy

W tym ćwiczeniu nauczysz się obliczać i wykorzystywać lokalne cechy obrazu za pomocą detektorów i deskryptorów cech.

Możesz zastosować detektory, korzystając z szeregu wbudowanych funkcji, jak w poniższym przykładzie (dla zastosowania detektora narożnego Harrisa).

```
points =
detectHarrisFeatures(im,'MinQuality',0.55,'FilterSize',15);
```

Jak widać, istnieje kilka parametrów, których wartości można zmienić. Wartości te można dostroić, aby uzyskać pożądane rezultaty, np. aby wykryto wszystkie punkty narożniki obecne w obrazie, ale nie znaleziono żadnych podwójnych lub potrójnych narożników w tym samym miejscu i żadnych narożników z powodu szumu obrazu. Przeczytaj dokumentację MATLAB, aby dowiedzieć się o znaczeniu wszystkich parametrów, a następnie spróbuj znaleźć wartości, dla których uzyskasz najlepszy wynik dla swojego obrazu.

Możesz również otrzymać tylko N narożników, dla których odpowiedź detektora będzie największa.

```
points = points.selectStrongest(N);
```

Teraz możesz wyświetlić narożniki znalezione przez algorytm, korzystając z następującego kodu:

```
points.Location = points.Location + [cropRect(1) cropRect(2)];

figure,
```

```
imshow(im);
hold on;
x = points.Location(:,1);
y = points.Location(:,2);

plot(x,y,'*g');
hold on
```

W kodzie zmienna cropRect jest dodawana, jeśli najpierw przyciąłeś obraz funkcją imcrop. Jeśli używasz pełnego obrazu, ustaw cropRect na wektor zerowy.

Aby zobaczyć, jak działa detektor Harrisa, zaimplementuj detektor w funkcji samodzielnie i sprawdź, jak działa w porównaniu z detektorem Harrisa zaimplementowanym w MATLAB-ie. Ćwiczenie będzie interesujące, ponieważ obejmuje wiele procesów przetwarzania obrazu, których nauczyłeś się do tej pory.

Przede wszystkim Twoja funkcja musi odczytać obraz i przekonwertować go na reprezentację w skali szarości. Następnie przekonwertuj swój obraz na double.

```
I = double(I);
```

Następnie musisz utworzyć miarę 'narożności '. Pierwszym krokiem jest uzyskanie obrazu gradientowego. Możesz również zastosować filtry, które obliczają pierwszą pochodną, takie jak filtr Sobela.

```
[Ix, Iy] = gradient(I); % Gradient in x and y direction
```

Teraz oblicz iloczyn pochodnych.

```
Ixx = Ix.^2;
Ixy = Ix .* Iy;
Iyy = Iy.^2;
```

Jak omawialiśmy w poprzednich ćwiczeniach, przydatne jest rozmycie obrazu przed dalszymi obliczeniami. Dlatego zastosuj filtrowanie Gaussa.

```
sigma = 1; % Standard deviation for Gaussian kernel
windowSize = 3; % Size of the window for Gaussian filter

G = fspecial('gaussian', windowSize, sigma);
Ixx = conv2(Ixx, G, 'same');
Ixy = conv2(Ixy, G, 'same');
Iyy = conv2(Iyy, G, 'same');
```

Obliczmy teraz funkcję odpowiedzi detektora Harrisa, korzystając ze stałej k podanej w literaturze.

```
k = 0.04; % Harris corner constant
R = (Ixx .* Iyy - Ixy.^2) - k * (Ixx + Iyy).^2;
```

Teraz, używając progu, znajdź tylko te punkty, dla których funkcja odpowiedzi detektora Harrisa była powyżej progu

```
threshold = 0.01 * max(R(:)); % Threshold is a fraction of the
maximum value of R
corners = R > threshold;
```

Teraz możesz zwizualizować narożniki, rysując je na obrazku.

Zadanie 3.6 Napisz program, który wykrywa narożniki na obrazie. Spróbuj wybrać parametry detektora narożników Harrisa, aby wykryć największą liczbę prawdziwych krawędzi. Spróbuj zminimalizować fałszywe wykrywanie, np. miejsca, w których nie ma narożników, krawędzi, unikaj podwójnych narożników w tych samych miejscach. Omów wyniki.

Zadanie 3.7 Powtórz zadanie 3.6, aby przetestować inne typy detektorów cech i deskryptorów dostępnych w MATLAB-ie. Zauważ, że niektóre z nich wykrywają cechy narożne, jak detektor Harrisa, inne są lepiej przystosowane do wykrywania cech typu blob. Otrzymasz różne obrazy zawierające oba te typy cech. Przetestuj, które detektory są lepsze do wykrywania danych typów, spróbuj znaleźć najlepszy zestaw parametrów dla danego obrazu, tak jak w poprzednim zadaniu.

W programie MALAB dostępne są następujące funkcje:

```
detectMinEigenFeatures();
detectFASTFeatures();
detectORBFeatures();
detectBRISKFeatures();
detectKAZEFeatures();
detectSURFFeatures();
```

Zastosowanie cech lokalnych – dopasowywanie cech

Jednym z najczęstszych zastosowań lokalnych cech obrazu jest dopasowywanie cech, w którym program dopasowuje tę samą cechę na wielu obrazach z tą samą sceną. Może być stosowany do wykrywania określonego obiektu na obrazie z wieloma obiektami i śledzenia obiektów w sekwencji wideo. Ponadto dopasowywanie cech jest jednym z głównych kroków we wszystkich algorytmach rekonstrukcji struktur 3D i ruchu. Jest ono używane do znajdowania odpowiadających sobie punktów, tj. projekcji tego samego punktu świata 3D na dwóch lub więcej obrazach uchwyconych przez system kamer. Wzajemna relacja odpowiadających sobie punktów jest konieczna do obliczenia głębi sceny. W przypadku zszywania obrazów dopasowywanie cech jest potrzebne do generowania obrazu panoramicznego.

Aby dopasować cechy w MATLAB-ie, nie wystarczy znać położenia punktów charakterystycznych na obrazie, ale najpierw należy wygenerować **wektor cech** dla każdego z tych punktów. Wektor cech zawiera informacje o opisie matematycznym rozkładu przestrzennego poziomów jasności i geometrii fragmentu obrazu w sąsiedztwie

tych punktów. Załóżmy, że mamy dwa obrazy zawierające tę samą scenę widzianą z dwóch punktów przestrzeni lub te same obiekty na dwóch różnych scenach i chcemy znaleźć zbiór odpowiadających sobie par punktów.

Dopasowywanie cech można przeprowadzić w następujący sposób:

Najpierw wczytaj dwa obrazki:

```
I1 = rgb2gray(imread('ImageLeft.png'));
I2 = rgb2gray(imread('ImageRight.png'));
```

Wyodrębniamy cechy za pomocą jednego z wcześniej analizowanych detektorów cech, np. detektora Harrisa

```
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
```

Możemy wyświetlić wykryte cechy, aby zobaczyć wyniki działania naszego programu. Następnie musimy zbudować reprezentację cech dla wszystkich wykrytych punktów cech.

```
[features1,valid_points1] = extractFeatures(I1,points1);
[features2,valid_points2] = extractFeatures(I2,points2);
```

Możemy dopasować cechy na podstawie ich reprezentacji matematycznej

```
indexPairs = matchFeatures(features1,features2);
```

Funkcja zwraca macierz Nx2 zawierającą pary indeksów odpowiadających sobie punktów. Następnie używamy tych indeksów, aby uzyskać współrzędne obrazowe tych par.

```
matchedPoints1 = valid_points1(indexPairs(:,1),:);
matchedPoints2 = valid_points2(indexPairs(:,2),:);
```

MATLAB udostępnia funkcję umożliwiającą szybką wizualizację zależności między punktami.

```
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,'montage
');
```

Zadanie 3.8 Wybierz detektor cech, który dał Ci najlepsze wyniki w poprzednim ćwiczeniu. Znajdź największą liczbę prawidłowych par odpowiadających sobie punktów w zestawach dwóch obrazów dostarczonych przez instruktora. Spróbuj powtórzyć ćwiczenie dla par obrazów uchwyconych przez Ciebie.

Zadania dodatkowe

Zadanie 3.9. Utwórz obraz zawierający kilka wzorców tego samego typu (wzorce mogą się nieznacznie różnić od siebie, możesz je modyfikować, dodając zakłócenia). Następnie napisz program MATLAB, który wykryje wszystkie wzorce danego typu i oznaczy je na obrazie. Program musi policzyć liczbę wzorców obecnych na obrazie. Przetestuj program na obrazach dostarczonych przez instruktora.

Zadanie 3.10. W tym ćwiczeniu dowiesz się, jak odporna jest korelacja na zmiany na obrazie.
Zobacz, jak algorytm wykrywania wzorców reaguje na następujące zakłócenia:
1) obraz jest rozjaśniony,
2) obraz jest przyciemniony,
2) obraz jest rozmyty przez filtr dolnoprzepustowy,
3) obraz jest zaszumiony (np. dodany szum „sól i pieprz", szum Gaussa itp.),
4) obraz jest obrócony (np. za pomocą funkcji MATLAB imrotate ()),
Przetestuj odporność funkcji korelacji, stopniowo zwiększając poziom każdego typu zakłócenia i znajdź wartość zakłócenia, dla którego funkcja nie znajduje wzorca. Sprawdź, czy skuteczność metody korelacji zależy od typu wykrywanego wzorca. Możesz pokazać swoje wyniki, rysując wykres pokazujący położenie środka wykrytego wzorca jako funkcję siły zakłócenia.

Zadanie 3.11 Proszę przetestować niezmienność cech na różne zmiany w obrazie. Przygotuj obiekt, który ma widoczny zestaw lokalnych cech. Dla co najmniej trzech dostępnych deskryptorów cech znajdź najlepsze parametry, aby wykryć największą liczbę prawdziwych lokalnych cech w obrazie odniesienia obiektu. Następnie zmień następujące elementy, robiąc zdjęcia obiektu po każdej zmianie: odległość aparatu od obiektu, ostrość na obiekcie (lekkie rozmycie), kąt widzenia aparatu względem obiektu, warunki oświetlenia. Po wprowadzeniu każdej ze zmian obserwuj, czy cechy są nadal wykrywane i czy są wykrywane w prawidłowym miejscu. Omów wyniki.