Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Podstawy sztucznej inteligencji i uczenia głębokiego
Kurs dla kierunku Inżynieria Mechatroniczna

Instrukcja 7:

Przetwarzanie obrazu: Segmentacja obrazu

Dowiesz się: czym jest segmentacja obrazu, jakie są podstawowe algorytmy segmentacji dla obrazów w skali szarości, progowanie, jak wdrożyć segmentację wielopoziomową za pomocą operacji logicznych na obrazach, jak segmentować obraz kolorowy w przestrzeniach kolorów RGB i HSV, czym są filtry tekstur i jak je stosować w segmentacji, czym jest algorytm K-Means w segmentacji obrazu, jakie operacje morfologiczne są stosowane do obrazów binarnych, czym są obiekty w obrazie binarnym i jak znaleźć cechy obiektów, które są stosowane w klasyfikacji obiektów.

Materiały dodatkowe:

- Wykład kursowy
- Baza obrazów

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

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

Segmentacja obrazu

Na ostatnich zajęciach widziałeś prosty algorytm segmentacji obrazu wykorzystujący operację progowania. Zacznijmy od przejrzenia koncepcji.

Progowanie (Thresholding) konwertuje obraz w skali szarości na obraz czarno-biały. Opiera się na progu binaryzacji. Na przykład piksele o wartości intensywności mniejszej niż próg otrzymują wartość 0, a piksele o wartości większej niż próg otrzymują wartość 1.

W programie MATLAB binaryzację można przeprowadzić za pomocą funkcji **`im2bw()`** lub operatora nierówności, jak omówiono w poprzednich instrukcjach laboratoryjnych.

Piksele obrazu binarnego można rozumieć jako zmienną logiczną (0 i 1). Dlatego też na macierzach obrazu binarnego można wykonać dowolną operację logiczną. Na przykład:

1) Iloczyn Logiczny - **`and(im1,im2);`**

2) Suma logiczna - **`or(im1,im2);`**

3) Negacja logiczna - **`not(im1)`**;

Operacje arytmetyczne są również wykonywane na obrazach (binarnych i w skali szarości):

1) Dodawanie - **`imadd(im1,im2);`**
2) Odejmowanie **- `imsubtract(im1, im2);`**
3) Mnożenie **- `immultiply(im1,im2);`**

Poniżej znajduje się przykładowy kod binaryzacji obrazu (z zastosowaniem dwóch metod):

```
clc;
clear all;
close all;

imRGB = imread('Crow.jpg');
imGray = rgb2gray(imRGB);

thresh1 = graythresh(imRGB); % an automatic threshold computation, try also your own
chosen thresholds

thresh2 = floor(thresh1*255); % determine threshold value in the range 0-255

imBW1 = im2bw(imGray,thresh1);

imBW2 = imGray > thresh2;

figure
subplot(1,2,1);
imshow(imBW1);
subplot(1,2,2);
imshow(imBW2);
```

Operacje morfologiczne na obrazach binarnych

Przetwarzanie morfologiczne wykorzystuje operacje logiczne do wprowadzania zmian w obrazach. Operacje te mogą być stosowane zarówno do obrazów binarnych, jak i w skali

szarości (jak omówiono w poprzednich ćwiczeniach laboratoryjnych). W ogólnym przypadku operacja morfologiczna wymaga przesunięcia maski, zwanej elementem strukturalnym, wzdłuż obrazu i określenia, dla każdej pozycji maski, wynikowej wartości operacji logicznej między pikselami maski a pikselami obrazu leżącymi bezpośrednio pod maską. Efekt najważniejszych operacji morfologicznych dla obrazów binarnych zostanie opisany poniżej.

Należy pamiętać, że w programie MATLAB **obiekty** mają **białe** piksele (logicznie jedynki), a piksele **tła** są **czarne** (logicznie zera).

Aby użyć filtra morfologicznego, należy utworzyć maskę (element strukturalny)

```
se1 = strel('type,size);
```

Gdzie 'type' oznacza typ elementu strukturyzującego, a size określa jego rozmiar. Po utworzeniu maski możesz zastosować jeden z wybranych filtrów morfologicznych. Podstawowe filtry to:

1) **Erozja** - filtr usuwa szum w postaci obiektów o rozmiarze mniejszym od rozmiaru elementu strukturalnego, zmniejsza powierzchnię wszystkich obiektów, rozłącza obiekty połączone cienkimi mostkami

   ```
   bw_erode = imerode(im_bw,se1);
   ```

2) **Dylatacja** - filtr usuwa szumy w postaci „dziur" w obiektach, zwiększa powierzchnię wszystkich obiektów, łączy obiekty znajdujące się blisko siebie

   ```
   bw_dilate = imdilate(im_bw,se1);
   ```

3) **Otwarcie** - jest połączeniem sukcesywnej erozji i dylatacji. Usuwa szum w postaci małych obiektów, ale nie zmienia powierzchni dużych obiektów.

   ```
   bw_open = imopen(im_bw,se1);
   ```

4) **Zamykanie** - filtr jest połączeniem sukcesywnego rozszerzania i erozji. Usuwa szum w postaci „dziur" w obiektach, ale nie zwiększa ich powierzchni.

   ```
   bw_close = imclose(im_bw,se1);
   ```

W programie MATLAB do stosowania filtrów morfologicznych można również użyć funkcji **bwmorph()**

Przykładowy kod pokazujący jak używać tych operacji:

```
clc;
clear all;
close all;

imRGB = imread('Crow.jpg');
imGray = rgb2gray(imRGB);

thresh = graythresh(imRGB); % an automatic threshold computation, try also your own
chosen thresholds
imBW = im2bw(imGray,thresh);
```

```
se = strel('square',5);

imErode = imErode(imBW,se);
imDilate = imdilate(imBW,se);
imOpen = imopen(imBW,se);
imClose = imclose(imBW,se);


figure,
subplot(2,2,1)
imshow(imErode);
title('Erosion')

subplot(2,2,2)
imshow(imDilate);
title('Dilation')

subplot(2,2,3)
imshow(imOpen);
title('Opening')

subplot(2,2,4)
imshow(imClose);
title('Closing')
```

Teraz przećwiczmy to, co do tej pory wiesz, na ćwiczeniach praktycznych

Zadanie 2.1 Napisz skrypt, wykonujący proste operacje morfologiczne na obrazach binarnych. Wykonaj progowanie obrazu, aby uzyskać obraz binarny z obecnymi na nim obiektami. Zastosuj cztery operacje morfologiczne: dylatację, erozję, otwarcie i zamknięcie. Zastosuj 3 różne rozmiary maski filtrów i 3 różne typy masek.

Podstawy analizy obrazu – wykrywanie obiektów

Głównym powodem, dla którego przeprowadzamy segmentację obrazu, jest uproszczenie obrazu do dalszej analizy obrazu. W obrazie binarnym obiekty są oddzielone od tła, co upraszcza obliczenia wszelkich cech powiązanych z obiektami.

Rozpoznawanie i klasyfikacja obiektów to jedne z najważniejszych zadań widzenia komputerowego. W laboratorium nauczysz się, jak obliczać cechy obrazu na podstawie obrazu binarnego. Aby przeanalizować i wykryć obiekt, obraz musi zostać najpierw przygotowany. Należy usunąć szum i wykonać konwersję skali szarości na binarną. Obiekty są reprezentowane przez piksele o wartości 1, a piksele tła mają wartość 0.

Najpierw wszystkie obiekty muszą zostać oznaczone za pomocą funkcji **`bwlabel()`** w MATLAB-ie. Ta funkcja po prostu nadaje numery (etykiety) wszystkim połączonym obiektom na obrazie.

```
[imLabel, num] = bwlabel(imGray, nb);
```

Funkcja przyjmuje binarny obraz i typ sąsiedztwa dla pikseli obiektu. Można używać sąsiedztw 4 i 8-pikselowych. Funkcja zwraca etykietowany obraz, w którym piksel ma przypisaną etykietę w zależności od przynależności do jakiegoś obiektu.

Następnie parametry geometryczne opisujące kształty są obliczane za pomocą funkcji **`regionprops()`**. Składnia funkcji jest następująca

```
features = regionprops(imLabel, properties);
```

gdzie zmienna **properties** jest ciągiem znaków opisującym, które cechy geometryczne chcesz uzyskać. Najprostszym wyborem właściwości są 'all' i 'basic'. Pierwsza z nich oznacza, że wszystkie cechy muszą zostać obliczone, a druga podaje tylko te podstawowe. Cechy są przechowywane w macierzy struktur MATLAB-a, która przechowuje wartości liczbowe wszystkich cech obliczonych dla wszystkich obiektów.

Oto prosty program, który oblicza podstawowe cechy.

```
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,'basic');
```

Zadanie 2.2 Uruchom kod do analizy obiektów binarnych. Wypróbuj go na obrazach z różnymi typami kształtów dostarczonych przez instruktora lub utworzonych w oprogramowaniu graficznym. Zamień obrazy na binarne, tak aby pozostały na nich obiekty i przeprowadź etykietowanie. Zobacz, jak wartości cech zmieniają się dla każdego obiektu.

Zadanie 2.3 Zbinaryzuj obraz dany przez instruktora, aby wyodrębnić z niego tylko jeden konkretny obiekt. Po binaryzacji musi pozostać tylko jeden biały obiekt, należy usunąć cały szum. Obiekt powinien być jak najbardziej zbliżony kształtem i rozmiarem do oryginału.
W przypadku większości obrazów nie wystarczy wybrać jednego progu binaryzacji. Może być konieczne napisanie funkcji, która wymaga określenia dwóch progów binaryzacji i zastosowania operacji logicznych.

Oto mały szablon kodu, który może okazać się przydatny przy rozwiązywaniu zadania 2.3

```
clc;
clear all;
close all;

imRGB = imread('YourImage.jpg');
imGray = rgb2gray(imRGB);

% Here, maybe you need to apply some preprocessing like filtering or morphological
operations applied to the grayscale image

% Here, you have to write your code for binarization, as described in the Task 2.1
```

```
% Here, you may apply additional filtering of binary image to get rid of the noise

% Diplay your results
figure,
subplot(1,2,1)
imshow(imGRay);
title('Original Image')

subplot(1,2,2)
imshow(myResult);
title('My Result')
```

Segmentacja oparta na teksturze

Segmentacja oparta na progach działa tylko wtedy, gdy obiekty i tło są mniej więcej jednolite pod względem jasności lub koloru. W przypadku obiektów teksturowanych ta metoda jest bardzo trudna do wdrożenia w praktyce i może prowadzić do obiektów o niewłaściwych kształtach. Filtry tekstur mogą być używane do wykrywania obszarów różniących się teksturą. W MATLAB-ie są trzy podstawowe filtry tekstur:

```
entropyfilt();
stdfilt();
rangefilt();
```

Każdy filtr oblicza pewną funkcję dla każdego piksela na obrazie, z wartościami zależnymi od rozkładu jasności pikseli w jego otoczeniu. Domyślnie sąsiedztwo pikseli wynosi 3x3, ale można je również określić jako parametr.

Na przykład filtr **`stdfilt()`** oblicza odchylenie standardowe pikseli otaczających analizowany piksel. Można go użyć do oddzielenia obszarów jednorodnych od obszarów, w których zmienność jasności jest duża. Każdy z tych filtrów upraszcza obraz, tak aby jego późniejsza segmentacja była możliwa przy użyciu standardowego progowania.

Obrazy uzyskane z filtrów tekstur mogą nie mieć wartości pikseli w standardowym zakresie obsługiwanym przez funkcję wyświetlania i przetwarzania obrazu. Dlatego przed dalszym przetwarzaniem może być konieczne znormalizowanie wartości pikseli do standardowego zakresu.

Możesz również zobaczyć wyniki tych filtrów bezpośrednio, jeśli używasz innej funkcji do wyświetlania obrazów niż standardowy **`imshow()`**. Na przykład możesz użyć poleceń **`image()`** lub **`imagesc()`**, które wyświetlają zawartość macierzy jako obraz, używając metody pseudokolorowania .

Zadanie 2.4. Napisz program do segmentacji obrazów tekstur dostarczonych przez instruktora. Pamiętaj, że różne filtry mogą działać najlepiej dla różnych obrazów i tekstur (dlatego są trzy do przetestowania).

Metoda segmentacji obrazu metodą K-Means

Metoda K-Means jest nienadzorowaną metodą klasyfikacji obiektów. Dzieli ona dane N-wymiarowe na K klastrów na podstawie K centroidów. Liczba klastrów K musi być znana z góry i jest podawana przez użytkownika. W metodzie tej dla segmentacji obrazów, wymiar przestrzeni cech zależy od liczby cech obliczonych dla pikseli obrazu. W standardowej

segmentacji obrazu kolorowego wektor cech jest trójwymiarowy i zawiera wartości 3 kanały kolorów. Dane są grupowane w K klastrów poprzez iteracyjne obliczanie K centroidów klastrów i obliczanie przynależności każdego punktu danych do klastra z najbliższym centroidem.

Segmentujmy obraz kolorowy za pomocą K-Means. Po załadowaniu obrazu kolorowego RGB do obszaru roboczego MATLAB zdecyduj, ile kanałów chcesz użyć do segmentacji.

`im_channels = im1(:,:,1:3);` - w tym przykładzie, wszystkie 3 kanały kolorów są używane. Później możesz zmienić liczbę kanałów i zobaczyć wyniki segmentacji. Byłoby to bardziej pomocne w przypadku innych przestrzeni kolorów, w których informacje o kolorze są oddzielone od informacji o intensywności.

Aby wykonać klasteryzację metodą K-Means, napisz kod:

```
features = im_channels;
[labels, cmap] = imsegkmeans(features,numClusters);
im_label = label2rgb(labels);
```

Macierz cech w tym przykładzie jest taka sama, jak nasze kanały obrazu kolorowego, ale później będzie zawierała więcej atrybutów pikseli w postaci innych cech.

Należy pamiętać, że do konwersji segmentowanego obrazu na RGB używamy funkcji **`label2rgb()`**, aby poprawić zobaczenie poszczególnych obszarów po segmentacji.

Zadanie 2.5 Napisz kod dla segmentacji obrazu kolorowego metodą K-Means. Następnie zmień liczbę kanałów kolorów (zmniejsz wymiar przestrzeni cech) i zobacz wynik segmentacji. Spróbuj segmentować różne obrazy, za każdym razem wybierając liczbę klastrów na podstawie zawartości obrazu. Następnie przekonwertuj obraz w RGB na inną przestrzeń kolorów (HSV, CIE L*a*b*) i powtórz ćwiczenie. Omów zastosowanie różnych przestrzeni kolorów w segmentacji obrazu kolorowego.

Następnie dodajemy więcej cech do naszej przestrzeni cech. Po pierwsze wiemy, że piksele należące do tego samego obiektu powinny być przestrzennie bliżej siebie. Dlatego położenie piksela na obrazie (współrzędne X, Y) może być użyte jako nowy atrybut. Najpierw przygotujmy dwie nowe macierze, które dla każdego piksela przechowują odpowiednio współrzędne X i Y piksela.

```
nrows = size(im1,1);
ncols = size(im1,2);
[X,Y] = meshgrid(1:ncols,1:nrows);
```

I rozszerz naszą macierz cech

```
features = cat(3,im_channels, X,Y);
```

Innym przydatną cechą, zwłaszcza w przypadku obrazów naturalnych, są cechy opisujące tekstury obrazu. Jednym z najczęściej używanych filtrów w analizie tekstur jest filtr Gabora, nazwany na cześć Dennisa Gabora (1900-1979), który jako pierwszy zaproponował go do przetwarzania sygnałów 1D. Odpowiedź filtra pokazuje, czy istnieje określona składowa częstotliwości w określonym kierunku w zlokalizowanym obszarze wokół analizowanego

piksela. Matematycznie odpowiedź impulsowa filtra Gabora jest falą sinusoidalną pomnożoną przez funkcję Gaussa. Na potrzeby analizy obrazu możemy zaprojektować bank filtrów Gabora, z których każdy jest dostrojony do wykrywania określonej częstotliwości przestrzennej w określonym kierunku obrazu.

Po pierwsze, filtr Gabora jest zdefiniowany w MATLAB-ie dla obrazów w skali szarości, dlatego musimy przygotować dla niego wersję naszego obrazu w skali szarości.

```
imGray = rgb2gray(imRGB);
```

Następnie musimy utworzyć bank filtrów Gabora. W przykładzie użyto następujących wartości częstotliwości przestrzennych i orientacji filtrów:

```
waveStep = 1;
orientStep = 45;
wavelength = 2.^(0:waveStep:5) * 3;
orientation = 0:orientStep:135;
g = gabor(wavelength,orientation);
```

Na koniec filtrujemy obraz, korzystając ze wszystkich dostępnych filtrów w banku filtrów

```
gaborResults = imgaborfilt(imGray,g);
```

Przed dalszym przetwarzaniem przydatne jest usunięcie szumu

```
for i = 1:length(g)
     sigma = 0.5*g(i).Wavelength;
gaborResults(:,:,i) = imgaussfilt(gaborResults(:,:,i),3*sigma);
end
```

W naszym przykładzie utworzono 24 nowe obrazy. Każdy z nich reprezentuje odpowiedź na daną instancję filtru z banku Gabora. Wszystkie z nich są używane w następnym kroku w celu powiększenia przestrzeni cech. Po zmianie przestrzeni cech wykonaj klasteryzację jeszcze raz.

```
features = cat(3,im_channels,X,Y,gaborResults);
[labels, cmap] = imsegkmeans(features,numClusters);
im_label = label2rgb(labels);
```

Możesz również używać wyłącznie odpowiedzi filtra Gabora jako elementów przestrzeni cech. Spróbuj przetestować różne możliwości. Wybór elementów przestrzeni cech zależy od obrazu, który chcesz analizować, i typów obiektów, które chcesz segmentować z tła. Podobnie jak w przypadku poprzednich algorytmów segmentacji, wyniki segmentacji mogą być przetwarzane dalej w celu znalezienia cech geometrycznych obiektów, jak zobaczysz w kolejnych częściach laboratorium.

Zadanie 2.6 Przeprowadź segmentację obrazu z tekstura dostarczonego przez prowadzącego. Po filtracji obrazów za pomocą filtrów Gabora zwizualizuj wyniki wszystkich filtrów. Spróbuj wybrać filtry, które zapewniły najlepszą odpowiedź, którą można wykorzystać w segmentacji obrazu. Spróbuj odrzucić filtry, które dostarczyły powtarzających się informacji. Powinno to zmniejszyć przestrzeń cech i czas obliczeniowy.

Zadania dodatkowe

Zadanie 2.7. Napisz program, który będzie przeprowadzał segmentację kolorowego obrazu dostarczonego przez instruktora nie używając metody K-Means. Pamiętaj, że oprócz standardowej reprezentacji obrazu kolorowego, czyli RGB, MATLAB oferuje inne modele kolorów. Zanim zaczniesz pisać kod, znajdź informacje o tym jak kolor Jest reprezentowane w tych spacje.

Zadanie 2.8 Wiedząc, jak działają filtry morfologiczne, napisz skrypt, który wykrywa krawędzie obiektów binarnych, używając tylko operacji morfologicznych i prostych operacji logicznych. To podejście nazywa się gradientem morfologicznym.

Zadanie 2.9 Spróbuj zaproponować inny cechy które mogą być pomocne w segmentacji obrazu. Znaleźć najlepszą segmentację dla kilku obrazów dostarczonych przez instruktora. Dla każdego obrazu złożoność może być inna. Spróbuj znaleźć najlepszą liczbę klastrów i najlepszy zestaw cech, który da Ci najlepszy wynik segmentacji dla danego obrazu.