Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Rozpoznawanie Obrazów

Semestr II, Informatyka

mgr inż. Marcin Skobel

2022

Laboratorium nr 6: Detekcja krawędzi i linii.

I. Zagadnienia teoretyczne

Wstęp

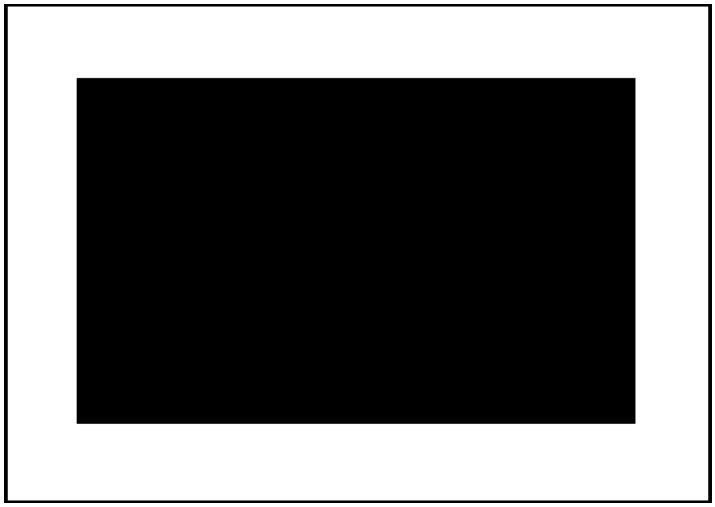
Detekcja krawędzi i linii stanowi niezwykle istotne zagadnienie w rozpoznawaniu obrazów. Linie i krawędzie często stanowią główny przedmiot analizy, wnosząc informację własnościach morfometrycznych badanych obiektów. Jako krawędź definiuje się miejsce na obrazie o znacznej zmianie wartości intensywności sąsiednich pikseli. Zatem jedną z najprostszych metod wykrywania krawędzi byłaby pochodna obrazu. Istnieją jednak nieco bardziej wyrafinowane techniki wykrywania linii i krawędzi które zostaną przedstawione w tym laboratorium.

Filtracja linii - Sobel i Prewitt

Detektor krawędzi Sobela polega na znalezieniu pikseli o dużej wartości gradientu. W przypadku tego filtra interesuje nas nie tylko sam fakt wystąpienia zmiany a także wielkość tego gradientu. Wielkość gradientu oblicza się, znajdując pierwiastek kwadratowy z sumy kwadratów pochodnej obrazu w kierunku x i pochodnej w kierunku y. Równanie gradientu w kierunku ψ może przyjąć następującą postać:

$$\bigtriangledown_{\psi} = \frac{\partial f}{\partial x} cos(\psi) + \frac{\partial f}{\partial y} sin(\psi), \tag{1}$$

gdzie x oznacza współrzędną na osi poziomej, y współrzędną na osi poziomej, ψ natomiast stanowi badany kierunek. Aby określić, jak duże wartości należy brać pod uwagę, ustalamy próg. Więc po znalezieniu gradientu bierzemy wszystkie wartości gradientu, które przekraczają ten konkretny próg. Rozważmy wypełnioną czarną ramkę (Rys. 1). Wszystkie piksele w czarnym polu mają podobne wartości pikseli, podczas gdy wartości pikseli na granicy lub krawędzi różnią się znacznie od sąsiednich pikseli. Dlatego warto rozważyć piksele z dużymi wartościami gradientu:



Rysunek 1: Przykładowy obrazek z wyraźnymi krawędziami

W filtrze Sobela dodatkowo wykorzystywane są następujące jądra:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2}$$

Jądro po lewej stronie wykrywa krawędzie poziome, a jądro po prawej stronie wykrywa krawędzie pionowe. W języku Python istnieje kilka bibliotek z gotowym filtrem Sobela.

Bardzo podobną metodą do podejścia Sobela jest metoda Prewitt. Zasadniczą różnicą są filtry zastosowane w metodzie Prewitt:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{3}$$

Podobnie jak w przypadku Sobela, suma współczynników w Prewitt również wynosi 0. Stąd filtr ten nie wpływa na piksele o stałej skali szarości. Filtr nie redukuje jednak szumów, co widać po wartościach współczynników.

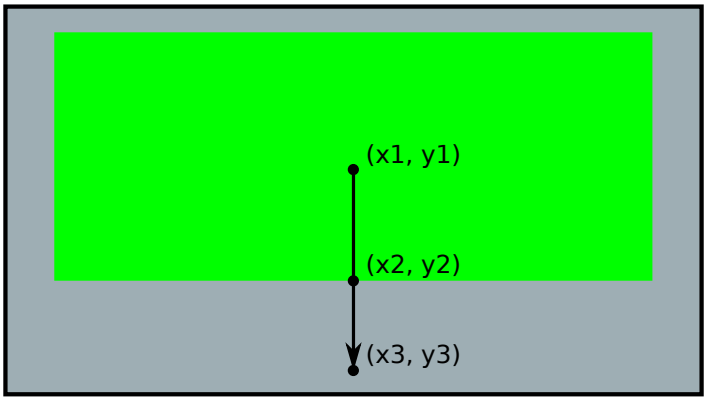
Filtracja linii - Canny

Drugim niezwykle istotnym podejściem do problemu wykrywania krawędzi jest metoda Canny. Wykorzystuje również koncepcję gradientów, podobnie jak w detektorze krawędzi Sobela, ale w poprzednim podejściu rozważaliśmy tylko wielkość gradientu. W podejściu Canny użyjemy również kierunku gradientu, aby znaleźć krawędzie. Algorytm Canny można podzielić na kilka kroków:

1. Wygładzanie obrazu za pomocą filtru Gaussa.

2. Znajdowanie gradientu: Następnym krokiem po usunięciu szumu jest znalezienie wielkości i kierunku gradientu poprzez obliczenie pochodnej x i pochodnej y. Kierunek jest ważny, ponieważ gradient jest zawsze prostopadły do krawędzi. Dlatego jeśli znamy kierunek gradientu, możemy również znaleźć kierunek krawędzi.

3. Tłumienie niemaksymalne: W tym kroku sprawdzamy, czy obliczony gradient jest maksymalnym spośród sąsiednich punktów leżących w dodatnim i ujemnym kierunku gradientu; to znaczy, czy są to lokalne maksima w kierunku gradientu. Jeśli nie jest to lokalne maksimum, to ten punkt nie jest częścią krawędzi. Na poniższym rysunku (Rys. 2) punkt (x2, y2) to lokalne maksimum, ponieważ ten punkt ma największą zmianę wartości pikseli i jest częścią krawędzi, podczas gdy pozostałe dwa punkty na linii nie mają dużej zmiany w wartościach pikseli i nie są lokalnymi maksimami:



Rysunek 2: Interpretacja maksimum lokalnego

4. Progowanie: w tym algorytmie używamy dwóch wartości progowych - wysokiego progu i niskiego progu, w przeciwieństwie do Sobel, gdzie użyliśmy tylko jednej wartości progowej. Nazywa się to progiem histerezy. Zrozummy, jak to działa. Wybieramy wszystkie punkty brzegowe, które są powyżej górnego progu, a następnie sprawdzamy, czy sąsiedzi tych punktów, którzy są poniżej wysokiego progu, ale powyżej niskiego progu; wtedy ci sąsiedzi również będą częścią tej krawędzi. Ale jeśli wszystkie punkty krawędzi znajdują się poniżej wysokiego progu, wówczas te punkty nie zostaną wybrane.

Transformata Hougha

Ostatnie pytanie brzmi czy istnieje jakaś prosta metoda wykrywania krawędzi nieliniowych takich jak elipsy, koła czy krzywe? W tym punkcie dowiemy się jak zastosować transformację Hougha do wykrywania krawędzi. . Transformacja Hougha to ogólna struktura, która przyjmuje sparametryzowane równania sztywnych kształtów i wykrywa te kształty na obrazie. Przyjrzymy się tylko liniom prostym i okręgom, ale technikę tę można rozszerzyć na dowolny inny kształt. Zacznijmy od wykrycia prostych linii, a następnie przejdźmy do okręgów.

W matematyce prosta jest definiowana za pomocą parametru stałej (punktu przecięcia z osią y) oraz nachylenia. Te dwa parametry można również ustalić dla prostych znajdujących się na obrazie. Stałą i nachylenie można obliczyć na podstawie dwóch dowolnych punktów w przestrzeni obrazu. Przykładowo, posiadając dwa punkty o współrzędnych $(x_1,\ y_1)$, $(x_2,\ y_2)$ równanie prostej można zdefiniować następująco:

$$y = ax + b \tag{4}$$

Dla $(x_1,\ y_1)$:

$$y_1 = ax_1 + b \tag{5}$$

Dla (x_2, y_2):

$$y_2 = ax_2 + b \tag{6}$$

Rozwiążemy te równania, aby znaleźć (a, b) i policzyć liczbę punktów, które to spełniają (a, b). Po uruchomieniu algorytmu na całym obrazie będziemy wiedzieć, ile punktów spełnia daną parę wartości (a, b). Nazwijmy te wartości wynikiem dla danej pary. Używając ręcznie ustawionego progu, wybieramy tylko te pary (a, b), które mają wynik większy niż próg. Algorytm zwraca te pary, a użytkownik może następnie narysować linię, używając nachylenia i stałej wartości. Warto tutaj zauważyć, że używając tylko nachylenia i stałej wartości, nigdy nie będziesz w stanie odgadnąć punktów końcowych linii i nigdy nie będziesz w stanie dokładnie narysować linii na obrazie. Aby rozwiązać ten problem, można wraz z wynikiem zachować listę punktów, które odpowiadają danej parze (a, b) i na ich podstawie obliczyć punkty końcowe. Istnieje inna odmiana linii Hougha, zwana probabilistyczną linią Hougha. Zasadniczo robi to samo, ale używa innego podejścia do obliczania parametrów linii. Wykorzystuje bardziej złożoną matematykę.

Wykrywanie krawędzi okręgów jest podobne do wykrywania linii z podstawową różnicą w postaci równania. Do wykrywania krawędzi okręgów należy zastosować równanie okręgu gdzie (a, b) jest środkiem okręgu a r jest promieniem:

$$(x - a)^2 + (y - b)^2 = r^2 \tag{7}$$

Teraz zamiast nachylenia i przecięcia, algorytm znajdzie współrzędne środka okręgu i jego promienia za pomocą punktów na obrazie.

II. Przykład praktyczny

Wstępna konfiguracja

```
import scipy.ndimage as ndimage
import cv2
import numpy as np
from google.colab import drive
drive.mount('/content/drive')
```

Wczytanie pliku

```
path = 'drive/My Drive/Colab Notebooks/Images/Test03.tif'
img = cv2.imread(path,0)
```

Metoda Sobel

```
sobelx = cv2.Sobel(img,0,1,0,ksize=3)
sobely = cv2.Sobel(img,0,0,1,ksize=3)
sobelxy = cv2.bitwise_or(sobelx,sobely)
plt.figure(figsize=(30, 21))
plt.subplot(4,1,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(4,1,2),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(4,1,3),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.subplot(4,1,4),plt.imshow(sobelxy,cmap = 'gray')
plt.title('Sobel XY'), plt.xticks([]), plt.yticks([])
plt.show()
```

Metoda Prewitt

```
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
prewittx = cv2.filter2D(img, -1, kernelx)
prewitty = cv2.filter2D(img, -1, kernely)
prewittxy = cv2.bitwise_or(prewittx,prewitty)
plt.figure(figsize=(30, 21))
plt.subplot(4,1,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(4,1,2),plt.imshow(prewittx,cmap = 'gray')
plt.title('Prewitt X'), plt.xticks([]), plt.yticks([])
plt.subplot(4,1,3),plt.imshow(prewitty,cmap = 'gray')
plt.title('Prewitt Y'), plt.xticks([]), plt.yticks([])
plt.subplot(4,1,4),plt.imshow(prewittxy,cmap = 'gray')
plt.title('Prewitt XY'), plt.xticks([]), plt.yticks([])
plt.show()
```

Metoda Canny

```
edges = cv2.Canny(img,50,300)
plt.figure(figsize=(20, 10))
plt.subplot(211),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(212),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

Transformacja Hougha - Detekcja okręgów

```
imgBlur = cv2.medianBlur(img,5)
cimg = cv2.cvtColor(imgBlur,cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(imgBlur,cv2.HOUGH_GRADIENT,1,20,param1=50,param2=30,
                           minRadius=0,maxRadius=50)
circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

plt.figure(figsize=(20, 10))
plt.imshow(cimg)
plt.show()
```

III. Uwagi

Do wykonania dzisiejszego przykładu warto pobrać plik testowy dostępny pod adresem:
`http://staff.uz.zgora.pl/mskobel/Test03.tif`
Plik z przykładami można pobrać ze strony: `http://staff.uz.zgora.pl/mskobel/lab9.py`

IV. Lista zadań

1. Na bazie skryptu Prewitt zmodyfikuj filtry tak aby uzyskały poniższą formę:

$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{8}$$

Które filtry lepiej wykrywają krawędzie jąder komórkowych?
2. Wykonaj detekcję okręgów za pomocą transformaty Hougha stosując obraz po wykrywaniu krawędzi Sobel XY jako obraz wejściowy
3.* Dla zainteresowanych: Na bazie przykładu zamieszczonego na stronie internetowej: `https://scikit-image.org/docs/dev/auto_examples/edges/plot_circular_elliptical_hough_transform.html` spróbuj na obrazie medycznym wykonać Transformację Hougha wykrywającą okręgi.
** Dla ekstremalnie zainteresowanych: wykonaj na obrazie medycznym Transformację Hougha wykrywającą elipsy.