Algorytm Canny'ego

Damian Wojtyczko

1 Wstęp

Operatory Sobela i Prewitta to matematyczne operatory różniczkowe, które są powszechnie stosowane w algorytmach przetwarzania obrazów i computer vision. Zastosowanie tych operatorów pozwala m.in. na wykrywanie krawędzi w obrazach, jednak wiąże się z pewnymi wadami (jak np. błędne wykrywanie szumu jako krawędzi). W tym artykule pokażę bardziej zaawansowany algorytm detekcji krawędzi, zwany algorytmem Canny'ego i opiszę jego kroki oraz podstawy matematyczne.

2 Wykrywanie krawędzi

Na początku, wyjaśnijmy czym w ogóle jest wykrywanie krawędzi. Wykrywanie krawędzi polega na znalezieniu w zadanym obrazie wszystkich krawędzi, czyli dokładniej granic pomiędzy dwoma obszarami (lub obiektami) w obrazie. Te zazwyczaj znacznie różnią się jasnością lub kolorem. Niemniej jednak, samo wykrywanie różnic w jasności lub kolorze może nie wystarczyć, ponieważ szum również cechuje się znaczącym odchyleniem tych wartości (a wykrywać szumu w tym przypadku nie chcemy). Dlatego też powstają różne algorytmy wykrywania krawędzi, które różnią się m.in. wrażliwością na szum i związanym też z tym czasem wykonywania (złożonością obliczeniową). Jednym z bardziej zaawansowanych (wieloetapowych) algorytmów jest właśnie Algorytm Canny'ego.

3 Algorytm Canny'ego (Canny algorithm)

Algorytm Canny'ego polega na wykonaniu następujących 5 kroków:

- Redukcja szumów filtrem Gaussa
- Obliczenie gradientu obrazu
- Supresja lokalnych nie-maksimów (Non-Maximum Suppresion)
- Podwójne progowanie (Double thresholding)
- Śledzenie krawędzi przez histerezę (Edge tracking by hysteresis)

3.1 Redukcja szumów

Polega ona na zastosowaniu filtru Gaussa na obrazie. Jest to operacja konwolucji obrazu (dla uproszczenia - w skali szarości) $O(x, y)$ z filtrem Gaussa:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Gdzie σ to współczynnik wygładzenia.
Operację konwoluncji (czyli zastosowania tego filtra) możemy zapisać następująco:

$$O_2(x, y) = O_1(x, y) * G(x, y)$$

W procesie redukcji szumów (czyli de facto wygładzania/rozmywania obrazu) zaczynamy od wygenerowania jądra Gaussa, czyli macierzy, której wartości obliczamy za pomocą funkcji Gaussa. W algorytmie Canny'ego najczęściej stosuje się macierz $5x5$. Następnie wykonujemy splot jądra Gaussa z naszym obrazem, czyli kolejno przesuwamy macierz po pikselach w obrazie i obliczamy sumę ważoną pikseli w tym obszarze. Na koniec otrzymujemy obraz z zredukowanym (rozmytym) szumem, co pozwoli nam na uniknięcie błędnego zakwalifikowania go jako krawędzi.

3.2 Obliczenie gradientu

W następnym kroku, obliczany jest gradient obrazu $O(x, y)$ w kierunku poziomym G_x i pionowym G_y. Najczęściej obliczany jest za pomocą operatorów dyskretnych, np. Sobela. W formie matematycznej, krok ten możemy zapisać następująco:

$$G_x = \frac{\partial O}{\partial x}, \quad G_y = \frac{\partial O}{\partial y}$$

W algorytmie wykorzystujemy operatory dyskretne w celu obliczenia powyższych pochodnych, zatem w przypadku operatora Sobela (najczęściej używanego w Algorytmie Canny'ego) możemy ją zapisać w postaci następujących macierzy:

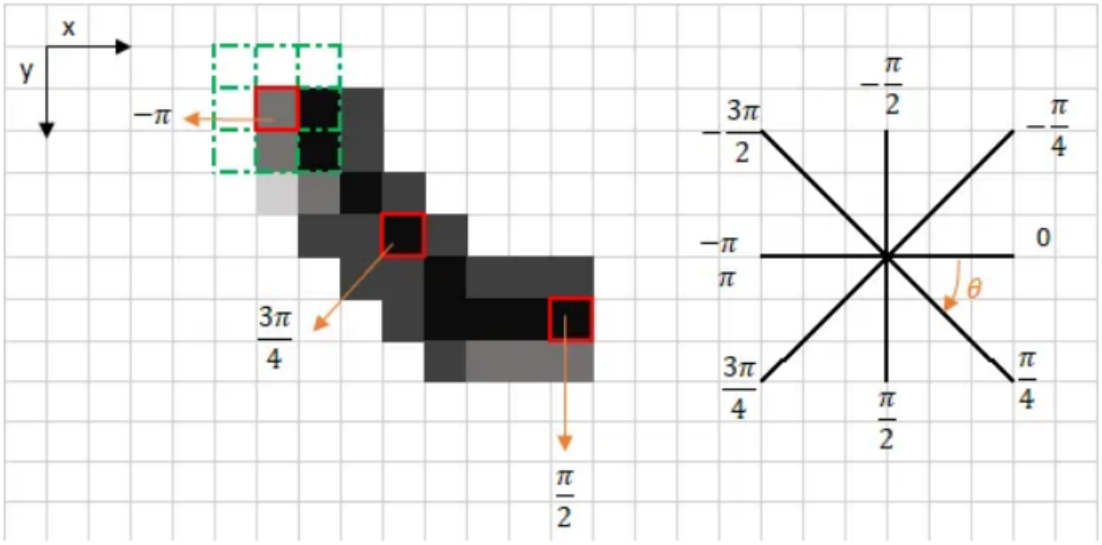
$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Następnie obliczana jest magnituda G oraz kierunek θ gradientu:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \arctan(\frac{G_y}{G_x})$$

3.3 Supresja lokalnych nie-maksimów (Non-maximum suppresion)

Kolejny krok polega na przenalizowaniu macierzy gradientu w celu wykrycia cienkich krawędzi. W tym kroku iteracyjnie dla każdego piksela (wartości w $O(x, y)$) sprawdzamy, czy jego wartość jest większa od sąsiednich pikseli wzdłuż kierunku θ. Jeśli tak, oznacza to, że jest to krawędź i zostawiamy wartość tego piksela bez zmian, a w przeciwnym przypadku ustawiamy wartość 0 (brak krawędzi). W celu ułatwienia zrozumienia tego kroku, posłużę się tutaj grafiką z artykułu Sofiane Sahira w ramach Towards Data Science (https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123):



3.4 Podwójne progowanie (Double thresholding)

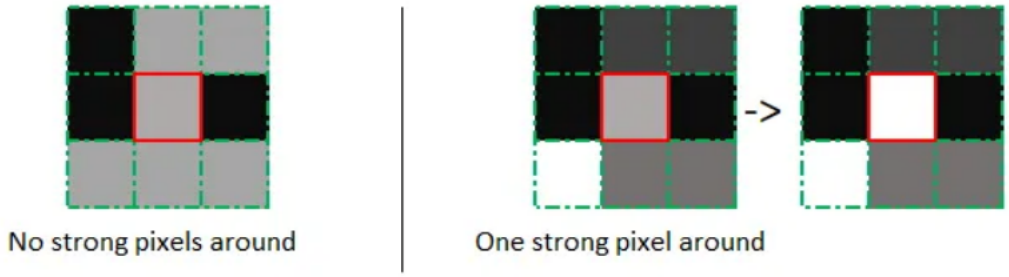
Następnie dokonujemy podwójnego progowania, czyli klasyfikujemy piksele do 3 grup:

- Silne - są to piksele, których wartość jest na tyle wysoka, że z pewnością możemy je uznać za krawędź.
- Słabe - są to piksele, których wartość nie jest bardzo wysoka, ale nie jest też niska - oznacza to, że mogą być krawędzią (ale nie muszą).
- Nieznaczące - są to piksele, których wartość jest niska i z pewnością nie są one krawędzią.

W celu zakwalifikowania pikseli do odpowiednich grup, ustalamy odpowiednie wartości (np. ustalając je jako procent względem najwyższych wartości w obrazie) dla 1. progu (pikseli słabych) i 2. progu (pikseli silnych). Piksele, które są na pograniczu dwóch progów kwalifikujemy do niższego (słabego) progu. Piksele, które nie trafią do żadnej z tych grup uznajemy za nieznaczące. Następnie wartość każdego piksela porównujemy z minimalnymi wartościami danego progu (wartościami od których piksel zostaje do danego progu zakwalifikowany) i we współrzędnych danego piksela ustawiamy wartość odpowiednią dla jego progu (dla pikseli nieznaczących wartość wynosi zawsze 0) i nadpisujemy obraz.

3.5 Śledzenie krawędzi przez histerezę (Edge tracking by hysteresis)

Na koniec, dokonujemy śledzenia krawędzi przez histerezę, która polega na zmianie kwalifikacji pikseli z 1. progu (pikseli słabych) na próg 2. (pikseli silnych), jeśli przynajmniej jeden z sąsiednich pikseli jest pikselem silnym. Ponownie, ilustracja z artykułu Sofiane Sahira pomaga lepiej zrozumieć tę ideę:

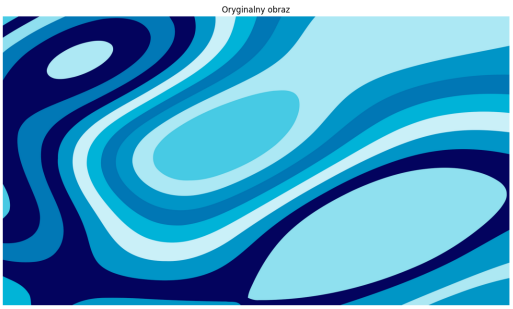


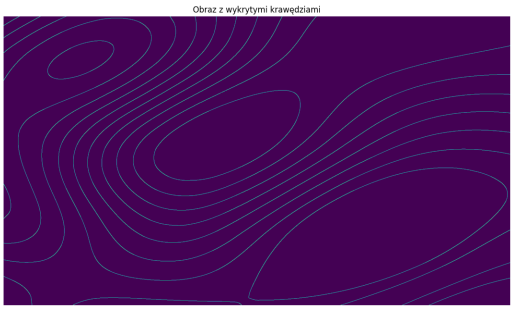
W wyniku działania algorytmu Canny'ego, otrzymujemy obraz zawierający jedynie krawędzie z pierwotnego obrazu. Wszystkie piksele krawędzi są tej samej jasności (wartości), a reszta obrazu jest czarna (wartość pikseli równa 0).

4 Przykład zastosowania algorytmu Canny'ego

W ramach przykładu, wykorzystamy funkcję `cv2.Canny()` z biblioteki OpenCV (cv2) w Pythonie. Jako wartość progu pikseli silnych przyjmiemy 100, a progu pikseli słabych wartość 20.

[ ]:
```
from matplotlib import pyplot as plt
import cv2

# Obraz Albert-Paul z https://pixabay.com/pl/illustrations/
  ↪wz%C3%B3r-projekt-scrapbooking-dekoracja-8798134/
original_image = cv2.imread("image.png")
original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
edges_image = cv2.Canny(original_image, 20, 100)

plt.figure(figsize=(30, 15))

plt.subplot(1, 2, 1)
plt.imshow(original_image)
plt.title("Oryginalny obraz")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(edges_image)
plt.title("Obraz z wykrytymi krawędziami")
plt.axis("off")
```





5 Zalety wieloetapowego wykrywania krawędzi

Algorytm Canny'ego dzięki zastosowaniu wielu różnych etapów posiada istotne zalety względem prostych jednoetapowych algorytmów:

- Niska wrażliwość na szum - dzięki zastosowaniu filtru Gaussa (który notabene jest narzędziem zupełnie matematycznym) redukujemy szum w obrazie, który mógłby zostać wykryty jako krawędź, co znacząco odróżnia ten algorytm od prostego wykrywania krawędzi np. samym operatorem Sobela.
- Niewielka ilość błędów - dzięki podwójnemu progowaniu i śledzeniu krawędzi przez histerezę znacząco redukujemy prawdopodobieństwo wykrycia fałszywych krawędzi.
- Dokładna lokalizacja krawędzi - dzięki supresji lokalnych nie-maksimów uzyskujemy w końcowym obrazie jedynie najważniejsze krawędzie, bez nieistotnych pikseli wokół krawędzi.

6 Źródła

Do napisania tego artykułu wykorzystano informacje z poniższych źródeł:

Materiały do wykładu z Analizy Matematycznej z Zastosowaniami 2
https://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm
https://medium.com/@abhisheksriram845/canny-edge-detection-explained-and-compared-with-opencv-in-python-57a161b4bd19
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
https://www.educative.io/answers/what-is-canny-edge-detection
https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html