



# Zadanie 02





## Kwadraty bez trójkątów (za to z wysokim stężeniem konwolucji)


W ramach tego zadania zaimplementujemy prosty mechanizm wykrywania kwadratów na fotografii. Wykorzystamy do tego pierwsze kroki detekcji krawędzi metodą Canny'ego oraz transformatę Hougha (po więcej informacji na temat obu technik sięgnij do PDFa z zagadnieniami i materiałami). Ponownie jednak - nie to jest naszym nadrzędnym celem. Najważniejsze jest to, by przy okazji nauczyć się implementować (w swoim ulubionym frameworku) różne rodzaje konwolucji oraz dowiedzieć się jak obserwować efekty ich stosowania.


Obie części zadania implementujemy z wykorzystaniem wybranego tydzień wcześniej frameworku. Każda ze wspomnianych metod ma w sieci wiele gotowych implementacji, ale znów - nie taki jest cel tego zadania.

- W przypadku punktów oznaczonych ikoną  poinformuj w jaki sposób je zrealizowałeś - wspomnij kluczowe klasy/metody/funkcje lub załącz powiązany fragment kodu źródłowego.
- W przypadku punktów oznaczonych ikoną  załącz w raporcie obraz przedstawiający stan kanału będącego wynikiem danej operacji.



### Najpierw poszukajmy krawędzi...

1. Zaczynamy od wczytania załączonego do zadania obrazu *furniture.jpg*. Tym razem nie ma potrzebny poddawania go na starcie intensywnej obróbce - upewnij się tylko, że można go użyć jako wejścia do zdefiniowanego dalej potoku przetwarzania. Warto jedynie przeskalować wszystkie wartości w pikselach tak, by mieściły się w zakresie od 0 do 1 (podobnie jak w poprzednim zadaniu - wystarczy podzielić je przez 255).
2. Obraz jest kolorowy i składa się z trzech kanałów - czerwonego, zielonego i niebieskiego. Dla uproszczenia późniejszych kroków zamieńmy go na wariant korzystający ze skali szarości - czyli składający się tylko z jednego kanału.
  1. W tym celu zastosuj konwolucję 1x1 z jednym kanałem wyjściowym. Przypisz odpowiednie wagi do wykorzystywanego filtra tak, by wynik był średnią ważoną wartości RGB dla każdego piksela (proporcje poszczególnych kolorów możesz dobrać "na oko" oceniając po prostu jakość wyniku). Dopilnuj by użyty był właściwy *padding* i *stride* - chcemy, by wynikowy obraz był tego samego rozmiaru co wejściowy. [ 
3. Obraz jest nieco za duży. Chociaż nie jest to praktyczne podejście do tego problemu, postaramy się zmniejszyć jego rozmiar z użyciem tzw. *poolingu* - operacji, która zastępuje każde okno o zadanym kształcie jednym nowym pikselem. W tym przypadku wystarczy nam okno 4 x 4. Pozostaje jeszcze pytanie o to, jakiego typu *pooling* wybrać - *average* czy *max*? Zdecyduj się na jeden z nich (i uzasadnij swój wybór). [ 
4. Obraz jest już mały i czarno-biały, teraz pora pozbyć się niepotrzebnych szumów i detali. W tym celu zastosujemy rozmycie gaussowskie.
  1. Zaczynaj od przygotowania kodu, który dla zadanego rozmiaru filtra  $n$  przygotuje tablicę  $n \times n$  zawierającą zgodne z rozkładem normalnym współczynniki - największe na środku, stopniowo malejące im bliżej krawędzi. Suma całej tablicy powinna oczywiście wynosić 1.

2. Następnie na zwróconym w poprzednim kroku obrazie zastosuj konwolucję  $n \times n$  wykorzystującą przygotowaną chwilę wcześniej macierz wag. Wyjściowy obraz powinien być zgodny z oryginałem co do rozmiaru - lecz estetycznie rozmyty. 
3. Poeksperymentuj z różnymi wartościami  $n$  - dobierz taki rozmiar filtra rozmywającego, który usunie drobne elementy, ale zachowa ogólny sens tego co na obrazie.
5. Kolejny etap to ustalenie, w których regionach obrazu gradient intensywności jest największy.

1. By poznać poziomą i pionową składową gradientu przetwórz (korzystając z konwolucji) aktualny stan obrazu przez wspomniane na zajęciach filtry Sobela. Zauważ, że tym razem na wyjściu uzyskujemy dwa kanały! 

```
[1, 2, 1   [1, 0, -1
0, 0, 0   2, 0, -2
-1, -2, -1  1, 0, -1]
```

2. Scal uzyskane kanały w jeden, który reprezentuje intensywność gradientu (czyli długość odpowiedniego wektora - obliczaną w zgodzie z normą Euklidesa). W tym celu kolejno. 
  1. Podnieś wszystkie wartości w obu warstwach do kwadratu.
  2. Zsumuj je - możesz użyć konwolucji  $1 \times 1$  lub dowolnej innej wygodnej metody.
  3. Wyciągnij pierwiastek z sumy.
3. Krawędzie to miejsca gwałtownych przejść tonalnych - można je rozpoznać po bardzo dużych gradientach. Aby pozostawić na obrazie krawędzie, odfiltruj regiony z niewielkim gradientem. 
  1. Można tu użyć np. tzw. progowanego ReLU (większość frameworków oferuje gotową implementację). Ta operacja sprowadza się w tym przypadku do "zachowaj wartość, jeżeli jest większa niż próg - w przeciwnym wypadku zastąp ją zerem".
  2. Poeksperymentuj z różnymi poziomami odcięcia i znajdź taki, dla którego znalezione krawędzie są tolerowalnej jakości.
  3. Na koniec znormalizuj wartości w kanale tak, by zawierały się w przedziale od 0 do 1.
6. To koniec części odpowiedzialnej za wykrywanie krawędzi. Jeżeli znasz metodę Canny'ego to na pewno zauważyłeś, że pominięto niektóre z jej istotnych kroków. Jeżeli chcesz, to możesz przetestować drugą połowę zadania z wyższej jakości krawędziami wykrytymi przez gotową implementację tego algorytmu - i sprawdzić na ile poprawią się finalne rezultaty - ale jest to zupełnie opcjonalne!

## ...a potem sprawdzimy, czy układają się w kwadratowe kształty.

6. Teraz wykorzystamy transformatę Hougha - każdy piksel musi zagłosować na możliwe położenia kwadratu, którego jest częścią. Środki takich kwadratów znajdują się...również na kwadracie. Dlatego głosowanie można przeprowadzić korzystając z operacji znanej jako transponowana konwolucja.
  1. Tak naprawdę nie wiemy jednak, jakiego rozmiaru są szukane przez nas kwadraty. Dlatego będziemy zmuszeni użyć nie jednej, a wielu konwolucji transponowanych - każda z nich reprezentować będzie inny rozmiar poszukiwanych kwadratów. Poniżej zaprezentowane są przykładowe filtry do wykorzystania w tym kroku (dla kwadratów o

rozmiarach 3x3 oraz 5x5). Wykorzystaj technikę by uzyskać przestrzeń głosowania dla wybranego zakresu rozmiarów (np. od 16x16 do 32x32). [🖼️🔍]

```
[1, 1, 1   [1, 1, 1, 1, 1
 1, 0, 1   1, 0, 0, 0, 1
 1, 1, 1]  1, 0, 0, 0, 1
           1, 0, 0, 0, 1
           1, 1, 1, 1, 1]
```

2. Jak wygląda przestrzeń głosowania dla każdego z rozważanych rozmiarów kwadratu? [🖼️🔍]
3. Maksima nowo powstałych kanałów reprezentują środki znalezionych kwadratów. Odfiltruj je w podobny sposób jak w przypadku krawędzi w poprzedniej sekcji (możesz pozwolić sobie na bardzo wysoki próg - interesują nas tylko najistotniejsze punkty, dosłownie kilka kropek - a i to tylko na niektórych kanałach). Pamiętaj, że próg warto dostosować do rozmiaru wykrywanego kwadratu - te większe mają szansę dostać proporcjonalnie więcej głosów.
4. Ponownie zastosuj konwolucję transponowaną, z tymi samymi kwadratowymi filtrami co poprzednio - teraz każda odkryta w poprzednim kroku kropka zostanie zastąpiona reprezentacją rzeczywiście wykrytego kwadratu (im mocniejsze maksimum, tym jaśniejszy taki kwadrat). [🖼️🔍]
5. Ta operacja może być dość kosztowna - na etapie implementacji zacznij od *proof of concept* wykorzystującego tylko fragment wejściowego obrazu. Jeśli wykonujesz zadanie na CPU, to możesz nawet na samym starcie zmniejszyć cały obraz dwukrotnie (i w ten sam sposób zredukować rozmiary filtrów powyżej).
7. Na koniec scal uzyskane wyniki z pierwotnym obrazem. Zwiększ ich rozdzielczość tak, by znów zgadzała się z oryginalną (korzystając z dowolnego zaimplementowanego już *upscalingu*), a następnie nałóż na startowy obraz, np. wzmacniając kanał zielony i osłabiając pozostałe - wtedy odnalezione kwadraty będą widoczne na końcowej fotografii. Czy ich pozycje mają sens? Wypróbuj swoją implementację na innym obrazie zawierającym kwadratowe elementy - czy efekty są podobnej jakości? [🖼️🔍]
8. Prawdopodobnie finalny efekt pozostawia sporo do życzenia - zapewne pojawiły się nietypowe artefakty, jedne kwadraty zostały wykryte wielokrotnie, inne zaś w ogóle. Jak myślisz, które etapy procedury miały najbardziej negatywny wpływ na wynik (jest to przecież wariant silnie okrojony)? Jak można by ją usprawnić (opowiedz, nie implementuj ;))?