Zadanie 25.04.2019

Operacje na obrazach bitmapowych

Matlab umożliwia import do swojej przestrzeni roboczej obrazów zapisanych w kilku formatach plików graficznych. Do tego celu można użyć polecenia:

```
Obraz1=imread('./sciezka/i/nazwa_pliku');
```

Tak przeczytany obraz reprezentowany jest w Matlabie jako macierz wartości, które odpowiadają kolorom punktów na obrazie (kolorom pikseli). W naszym ćwiczeniu dla uproszczenia będziemy posługiwać się obrazami w odcieniach szarości, więc wartości w macierzy odpowiadają po prostu stopniowi jasności punktów.
Wczytany obraz można wyświetlić:

```
image(Obraz1);
```

Najczęściej uzyskany na ekranie obraz, nie będzie odpowiadał kolorystyce zdjęcia źródłowego, w tym celu (w naszym przypadku) trzeba ustalić jeszcze tzw. mapę kolorów:

```
colormap gray;
```

Aby sprawdzić w jaki sposób Matlab interpretuje wartości zawarte w wyświetlanej macierzy można włączyć wyświetlanie skali kolorów:

```
colorbar;
```

Ponieważ standardowa mapa kolorów Matlaba zawiera 64 wartości (odcienie koloru) a nasze zdjęcia zakodowane są w skali 8 bitowej (czyli w zakresie 0 – 255) więc aby dobrze oddać kolorystykę trzeba wartości naszej macierzy obrazu podzielić na 4, np.:

```
image(Obraz1/4);
```

Teraz można poeksperymentować z innymi mapami kolorów, dostępne „gotowe” mapy to: parula, jet, hsv, hot, cool, spring, summer, autumn, winter, bone, copper, pink.

Do dalszych operacji należy zmienić typ wartości elementów naszej macierzy, gdyż najczęściej jest to typ uint8 i operacje zmiennoprzecinkowe, albo dające w wyniku liczby ujemne nie będą prawidłowo wykonywane (co można sprawdzić). Aby dokonać konwersji należy:

```
Obraz_po_konwersji=double(Obraz1);
```

Teraz można wykonać szereg prób z obrazem (najlepiej na mapie ‘gray’):
1) zmienić jasność (już właściwie osiągnięte wcześniej przez dzielenie wartości) i sprawdzić jak wpływa to na histogram obrazu (polecenie: `histogram(Obraz1)`)
2) odwrócić kolory (zrobić „negatyw” zdjęcia)
3) dokonać binaryzacji obrazu (wszystkie punkty powyżej pewnej jasności zrobić białe, a pozostałe czarne) – również można zobaczyć histogram
4) obliczyć różnicę obrazów – dzięki czemu daje się łatwo wskazać różnice w obrazach (na podobnych zasadach możne działać prosty detektor ruchu). Należy oczywiście w tym celu zaimportować więcej niż jeden obraz.

Bardziej zaawansowanymi operacjami są przekształcenia kontekstowe i konwolucja. Operacje tego typu można wykorzystać m.inn. do detekcji krawędzi. Szersze informacje na ten temat można znaleźć w [1]. Operacje kontekstowe to takie, w których dla uzyskania wartości jednego punktu obrazu docelowego bierze się pod uwagę większą ilość punktów obrazu źródłowego (najczęściej bezpośrednio ze sobą sąsiadujących) – strona 43 w [1].

Do detekcji krawędzi można wypróbować kilka filtrów, np. filtr Sobela którego maska wygląda tak:

-1	0	1
-2	0	2
-1	0	1

Algorytm działania jest następujący: aby wyznaczyć wartość punktu docelowego w obrazie należy wartości jego sąsiadów przemnożyć przez wartości filtra (maski) i zsumować je ze sobą. Położenie punktu obliczanego względem maski symbolizuje szary kolor na rysunku powyżej.

Poniżej kod w Matlabie:

```
Filtr= [ -1 0 1; -2 0 2; -1 0 1]
[X,Y]=size(ImZrodla);
ImZrodla=double(ImZrodla);
ImWynikowe=zeros(X,Y);
for x=[2:(X-1)]
    for y= [2:(Y-1)]
    ImWynikowe(x,y)=
      sum(sum(ImZrodla([x-1 : x+1], [y-1 : y+1]).*Filtr));
    end
end
```

Proszę przetestować działanie także dla innych filtrów (filtrów o innej masce).

Jako wynik tego ćwiczenia, można przedstawić skrypt wykonujący po kolei omawiane wyżej procedury.

Literatura:

[1] http://www.focus.agh.edu.pl/theses/MGR04.pdf