Analiza i przetwarzanie obrazu laboratorium 1

Marcin TORGiren Fabrykowski

2 marca 2013

1 Wstęp teoretyczny

1.1 Wyjaśnienie podstawowych pojęć

Piksel najmniejszy widzialny element obrazu na ekranie monitora. W systemie RGB składa się z trzech subpikseli o kolorach: czerwonym (Red), zielonym (Green) oraz niebieskim (Blue). W zależności od natężenia światła emitowanego przez pojedynczy subpiksel, wypadkowa barwa piksela zmienia się na zasadzie syntezy addytywnej.

Szum zakłócenia na obrazie. Jest on skutkiem niedoskonałości technologii pobierającej obraz ze świata zewnętrznego (różnice w czułości sąsiadujących elementów światłoczułych na matrycy). Jest on najbardziej widoczny gdy obraz powstał w warunkach niedoboru światła.

Szum sztuczny szum wygenerowany w sposób pseudolosowy za pomocą komputera.

Obraz ostry przenosi dużą ilość informacji (często nadmiarowych), wszystkie detale obiektu są wyraźne i łatwe do rozróżnienia.

Obraz słaby obraz zaszumiony, nieostry, rozmazany lub zmodyfikowany w inny sposób utrudniający jego odczytanie. Często przenosi on zbyt mało informacji do poprawnej klasyfikacji.

Wstępne przetwarzanie operacje przeprowadzane na obrazie wejściowym, w celu usunięcia zanieczyszczeń i przygotowania do ekstrakcji cech.

Projekcja (pozioma/pionowa) zliczenie pikseli w liniach lub kolumnach i przedstawienie zależności jako wykresu (histogramu).

Histogram wykres słupkowy, przedstawiający zależność wartości cechy od jej ilości. W analizie i przetwarzaniu obrazów histogram oznacza w szczególności wykres przedstawiający zależność wartości pikseli od ich ilości na obrazie. Oś pozioma zawiera rosnąco wartości pikseli od 0 do 255, natomiast oś pionowa - ilość wystąpień piksela o danej wartości barwy. W przypadku histogramu RGB każdy kanał rozpatrywany jest osobno.

1.2 Przetwarzany obraz

Obraz źródłowy znajduje się na rys. 1

Rysunek 1: Obraz źródłowy

2 Analiza obrazu

2.1 Negatyw

2.1.1 Kod funkcji

```
@image_loaded
def negative(self):
    """Tworzy negatyw obrazu"""
    data = np.array(self.__image.getdata())
    data = 255 - data
    data = [tuple(x) for x in data]
    self.__image.putdata(data)
```

2.2 Przetworzony obraz

Przetworzony obraz jest widoczny na rys. 2

Rysunek 2: Negatyw

2.3 Odcienie szarości

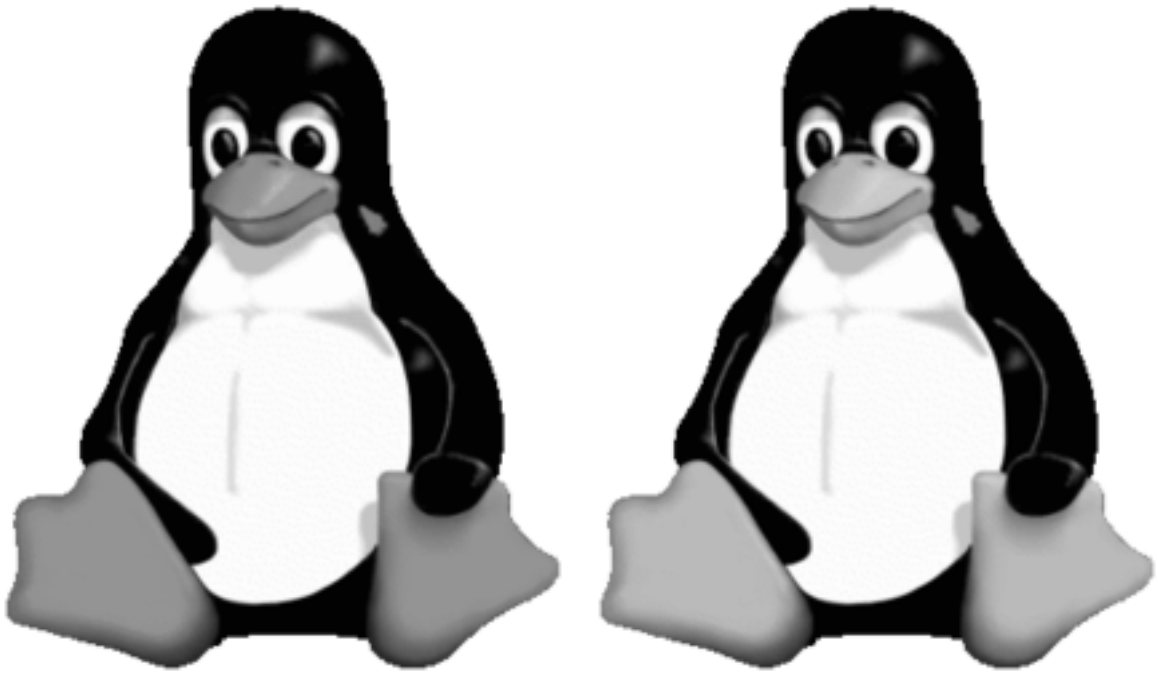
2.3.1 Kod funkcji

```
def __grayscale1(self):
    """Konwersja do skali szarosci"""
    data = np.array(self.__image.getdata())
    data = [3 * (int(x.mean()),) for x in data]
    self.__image.putdata(data)

def __grayscale2(self):
    """konwersja do skali szarosci"""
    data = np.array(self.__image.getdata())
    data = [3 * (int(0.3 * x[0] + 0.59 * x[1] + 0.11 * x[2]),)
                    for x in data]
    self.__image.putdata(data)
```

2.3.2 Przetworzony obraz

Przetworzony obraz jest widoczny na rys. 3



Rysunek 3: Obrazy w skali szarości

2.4 Normalizacja histogramu

2.4.1 Kod funkcji

```
@image_loaded
def normalize(self):
    data = np.array(self.__image.getdata())
    R = data[:, 0]
    G = data[:, 1]
    B = data[:, 2]
    R = (R - R.min()) * 255. / R.max()
    B = (B - B.min()) * 255. / B.max()
    B = (B - B.min()) * 255. / B.max()

    data[:, 0] = R
    data[:, 1] = G
    data[:, 2] = B

    data = [tuple(x) for x in data]
    self.__image.putdata(data)
```

2.4.2 Przetworzony obraz

Przetworzony obraz jest widoczny na rys. 4

Rysunek 4: Obraz ze znormalizowanym histogramem

3 Wnioski

- Druga metoda konwersji do odcieni szarości daje lepszy wynik wizualny, ponieważ ludzkie oko jest bardziej czułe na kolor zielony, co zostało uwzględnione we wzorze. W przypadku pierwszej metody, wszystkie składowe mają taką samą wagę.
- Wykonanie normalizacji histogramu pozwala na zwiększenie kontrastu obrazu.