AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Analiza i przetwarzanie obrazów

Ćwiczenie 3

Krystian Wojtas

Kraków, listopad 2011

1 Wstęp

Celem ćwiczeń była zmiana jasności i konrastu obrazu, zastosowanie filtrów wykrywania krawędzi - krzyż Robertsa oraz filtr Sobela, a także obrót o zadany kąt. Wykorzystany został język Ruby i jego framework RMagic, który binduje funkcje biblioteczne z pakietu ImageMagick.

Rysunek 1: Obraz wzorcowy

2 Jasność

Aby zmienić jasność pikseli, zastosowany został wzór

$$p(x, y) = p(x, y) + w$$

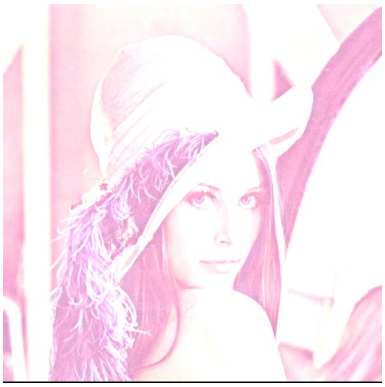
gdzie w jest wartością o jaką przesuwamy barwy w poszczególnych kanałach.

Listing 1: Implementacja zmiany jasnosci

```
1 def jasnosc(orginal,w)
2   edit(orginal) do |i, j, chb|
3     cut( chb[i][j] + w )
4   end
5 end
```

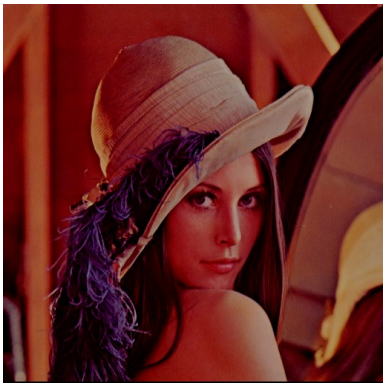
Rysunek 2: przesuniecie 0.1 Q

Rysunek 3: przesuniecie 0.3 Q

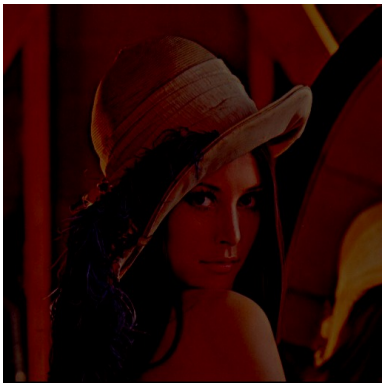


Rysunek 4: przesuniecie 0.6 Q

Rysunek 5: przesuniecie -0.1 Q



Rysunek 6: przesuniecie -0.3 Q



Rysunek 7: przesuniecie -0.6 Q

3 Kontrast

Aby zmienić kontrast pikseli, zastosowany został wzór

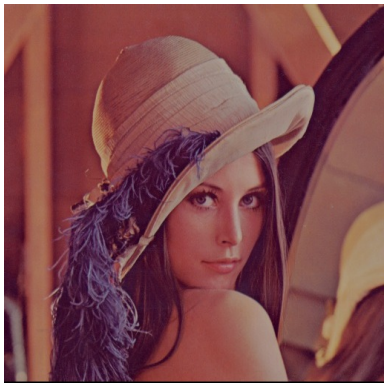
$$p(x, y) = p(x, y) * w$$

gdzie w jest wartością o jaką skalujemy barwy w poszczególnych kanałach.

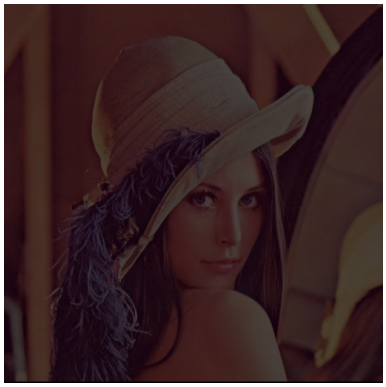
Listing 2: Implementacja zmiany kontrastu

```
1 def kontrast(orginal,w)
2   edit(orginal) do |i, j, chb|
3     cut( chb[i][j] * w )
4   end
5 end
```

Rysunek 8: kontrast 0.9

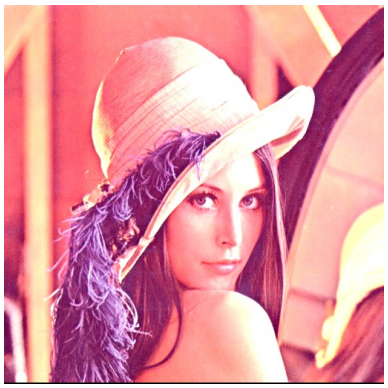


Rysunek 9: kontrast 0.7

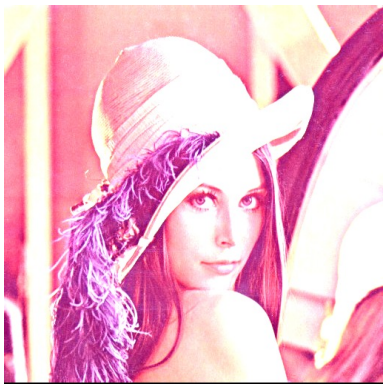


Rysunek 10: kontrast 0.3

Rysunek 11: kontrast 1.1



Rysunek 12: kontrast 1.5



Rysunek 13: kontrast 2.1

4 Krzyż Robertsa

Filtr znajduje krawędzie na obrazie. Działa według wzoru

$$p(x,y) = |p(x,y) - p(x+1,y+1)| + |p(x+1,y) - p(x,y+1)|$$

Listing 3: Implementacja krzyza Robertsa

```
1 def krzyzRobertsa(orginal)
2   edit(orginal, :top => 1, :bottom => 1, :left => 1, :right => 1) do |i, j, chb|
3     cut( (chb[i][j] - chb[i+1][j+1]).abs + (chb[i][j+1] - chb[i+1][j]).abs )
4   end
5 end
```

Rysunek 14: Zastosowany krzyz Robertsa

5 Filtr Sobela

Filtr również znajduje krawędzie na obrazie. Dla tak nazwanych pikseli otaczających piksel środkowy px

$$\begin{pmatrix} p0 & p1 & p2 \\ p7 & px & p3 \\ p6 & p5 & p4 \end{pmatrix}$$

zastosowanie mają wzory wyliczające nową wartość środkowego piksela

$$x = (p2 + 2 * p3 + p4) - (p0 + 2 * p7 + p6)$$

$$y = (p6 + 2 * p5 + p4) - (p0 + 2 * p1 + p2)$$

$$px = \sqrt{x^2 + y^2}$$

Listing 4: Implementacja filtru Sobela

```
1 def sobel(orginal)
2   edit(orginal, :top => 1, :bottom => 1, :left => 1, :right => 1) do |c, r, chb|
3     x = chb[c+1][r-1] + 2*chb[c+1][r] + chb[c+1][r+1]
4     y = chb[c-1][r+1] + 2*chb[c][r-1] + chb[c+1][r+1] - chb[c-1][r-1] - 2*chb[c
          ][r-1] - chb[c-1][r+1]
5     cut( Math.sqrt(x*x + y*y) )
6   end
7 end
```

Rysunek 15: Zastosowany filtr Sobela

6 Obrót

Przed przystąpieniem do obracania obrazka, należy ustalić punkty w które trafiłyby jego rogi. Wypadną one poza ramy obrazka - to nasuwa informację o korygowanym przesunięciu oraz nowych rozmiarach. Obrót obrazka uzyskujemy przekształcając położenie jego pikseli według wzorów na obrót punktu

$$x' = cos(r) * (x + p) - sin(r) * (y + q)$$

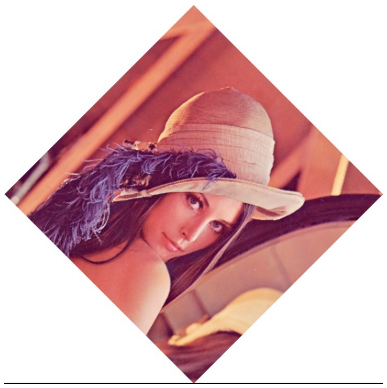
$$y' = sin(r) * (x + p) + cos(r) * (y + q)$$

$$p'(x, y) = p(x', y')$$

gdzie p i q to położenie lewego górnego rogu po obrocie oraz wektor przesunięcia wszystkich pikseli. Dzięki temu zaczynają się od (0,0).

Listing 5: Implementacja obrotu

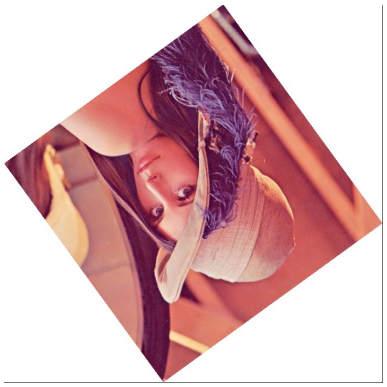
```
1  def obroc(i, j, kat)
2    [
3      ( Math.cos(kat)*i-Math.sin(kat)*j ).to_i,
4      ( Math.sin(kat)*i+Math.cos(kat)*j ).to_i,
5    ]
6  end
7
8  def obrot(orginal, kat)
9    #powrot jesli nie trzeba obracac
10   if kat == 0
11     return orginal
12   end
13   #ustalenie wymiarow obrazka i przesuniecia
14   xs = []
15   ys = []
16   [ [0, 0], [orginal.columns, 0], [orginal.columns, orginal.rows], [0, orginal.
       rows] ].each do |m|
17     px = obroc(m[0], m[1], kat)
18     xs.push( px[0] )
19     ys.push( px[1] )
20   end
21   xs.sort!()
22   ys.sort!()
23   #szerokosc i wysokosc nowego obrazka
24   width = xs[3] - xs[0]
25   height = ys[3] - ys[0]
26   #wektor przesuniecia [p, q]
27   p = xs[0]
28   q = ys[0]
29
30   edit(orginal, :columns => width, :rows => height ) do |i, j, chb|
31     px = obroc(i+q, j+p, kat)
32     if px[0] < orginal.columns and px[1] < orginal.rows and px[0] > 0 and px[1]
         > 0
33       cut( chb[px[0]][px[1]] )
34     else
35       Magick::QuantumRange
36     end
37   end
38 end
```



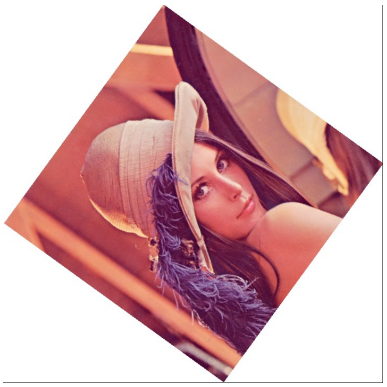
Rysunek 16: obrot 0.25 Pi

Rysunek 17: obrot 1.5 Pi

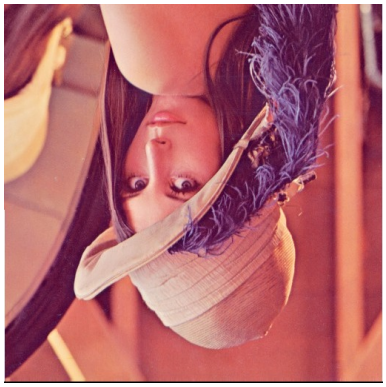
Rysunek 18: obrot 2.1 Pi



Rysunek 19: obrot 0.8 Pi



Rysunek 20: obrot -0.3 Pi



Rysunek 21: obrot Pi

Obraz obrócony o kąt nie będący wielokrotnością pi/2 jest rozmiarowo większy. Zawiera więc więcej pikseli niż orginał. Puste przestrzenie uzupełniłem czernią.