Analiza i Przetwarzanie Obrazów **Nr zajęć: 04** **Student: Marek Zyśko**	**13.11.2011 r.** **Prowadzący: mgr inż. Adam Szczepański**

I. **Teoria**

Na zajęciach nr 04 przyjrzeliśmy się metodom odszumiania obrazów. Początkowo wprowadziliśmy kilka rodzajów szumu sztucznego, aby następnie przetestować trzy metody odszumiania obrazu.

Wprowadzone przeze mnie zostały szumy typu:

Sól i pieprz – losowe umieszczenie pikseli białych i czarnych z pewnym założonym prawdopodobieństwem wystąpienia szumu w pikselu.

Szum równomierny – wprowadzenie w losowych miejscach na obrazie z pewnym prawdopodobieństwem zaszumienia w postaci zmiany wartości piksela o losową wartość z przedziału [-30; -21] lub [21; 30]. Szumem równomiernym2 nazywać będę tutaj ten sam rodzaj szumu jednak operacja zaszumiania w tym wypadku została przeprowadzona dla każdego kanału osobno.

Filtr medianowy – rodzaj filtru pozwalający zmniejszyć zakłócenia poprzez zastąpienie piksela środkową wartością z posortowanej listy wartości z określonego otoczenia badanego piksela.

Filtr medianowy2 – ten sam rodzaj filtru jednak z zastrzeżeniem, że aby zmiana została dokonana to wartość badanego piksela powinna znajdować się w 20% skrajnych wartościach.

Filtr za pomocą średniej – sposób działania jest podobny, ale w tym wypadku obliczana jest średnia wartość piksela ze wszystkich wartości z otoczenia badanego piksela i zastępowana jest wyliczoną wartością.

Źródłami szumów mogą być różne czynniki. Przykładowo powodem wystąpienia szumu może być uszkodzony lub zabrudzony obiektyw. Brak odpowiednio mocnego światła przy wykonywaniu zdjęcia też może być powodem pojawienia się szumu.

Z punktu widzenia analizy obrazu odszumianie jest bardzo istotne. Jeżeli szum na obrazie jest zbyt duży źle mogą zostać rozpoznane cechy charakterystyczne – to znaczy, że szum zostanie rozpoznany jako cecha charakterystyczna.

II. Kod programu

a) Szum typu sól i pieprz

```
private static BufferedImage szum_sol_pieprz(BufferedImage img, double p){
    BufferedImage szu=new BufferedImage(img.getWidth(), img.getHeight(), img.getType());
    for(int i=0;i<img.getWidth();i++)
        for(int j=0;j<img.getHeight();j++)
        {
            if(Math.random()>=p){
                if(Math.random()>=0.5)
                    szu.setRGB(i, j, toRGB(255,255,255));
                else
                    szu.setRGB(i, j, toRGB(0,0,0));
                    }
                else szu.setRGB(i, j, toRGB(getR(img.getRGB(i, j)),getG(img.getRGB(i, j)),getB(img.getRGB(i, j))));
        }
    return szu;
}
```

Z prawdopodobieństwem 1-p wprowadza czarny lub biały piksel na obrazie.

b) Szum równomierny

```
private static BufferedImage szum_rownomierny(BufferedImage img, double p){
    BufferedImage szu=new BufferedImage(img.getWidth(), img.getHeight(), img.getType());
    int r,g,b;
    for(int i=0;i<img.getWidth();i++)
        for(int j=0;j<img.getHeight();j++)
        {
            if(Math.random()>=p){
                int z = (int)(10*Math.random())+21;
                if(Math.random()>=0.5) z=-z;
                r=getR(img.getRGB(i, j))+z;
                g=getG(img.getRGB(i, j))+z;
                b=getB(img.getRGB(i, j))+z;
                if (r>255) r=255; if (r<0) r=0;
                if (g>255) g=255; if (g<0) g=0;
                if (b>255) b=255; if (b<0) b=0;
                    szu.setRGB(i, j, toRGB(r,g,b));
                    }
                else szu.setRGB(i, j, toRGB(getR(img.getRGB(i, j)),getG(img.getRGB(i, j)),getB(img.getRGB(i, j))));
        }
    return szu;
}
```

Z prawdopodobieństwem 1-p wprowadza zakłócenie o losową wartość z przedziału [-30; -21] lub [21; 30] dla każdego kanału RGB ta sama wartość.

c) Szum równomierny2

```
private static BufferedImage szum_rownomierny2(BufferedImage img, double p){
    BufferedImage szu=new BufferedImage(img.getWidth(), img.getHeight(), img.getType());
    int r,g,b;
    for(int i=0;i<img.getWidth();i++)
            for(int j=0;j<img.getHeight();j++)
            {
                if(Math.random()>=p){
                        int z1 = (int)(10*Math.random())+21;
                        int z2 = (int)(10*Math.random())+21;
                        int z3 = (int)(10*Math.random())+21;
                    if(Math.random()>=0.5) z1=-z1;
                    if(Math.random()>=0.5) z2=-z2;
                    if(Math.random()>=0.5) z3=-z3;
                    r=getR(img.getRGB(i, j))+z1;
                    g=getG(img.getRGB(i, j))+z2;
                    b=getB(img.getRGB(i, j))+z3;
                    if (r>255) r=255; if (r<0) r=0;
                    if (g>255) g=255; if (g<0) g=0;
                    if (b>255) b=255; if (b<0) b=0;
                        szu.setRGB(i, j, toRGB(r,g,b));
                        }
                    else szu.setRGB(i, j, toRGB(getR(img.getRGB(i,
j)),getG(img.getRGB(i, j)),getB(img.getRGB(i, j))));
            }
    return szu;
}
```

Z prawdopodobieństwem 1-p wprowadza zakłócenie o losową wartość z przedziału [-30; -21] lub [21; 30] dla każdego kanału RGB inna losowa wartość.

d) Filtr medianowy

```
private static BufferedImage filtr_medianowy(BufferedImage img, int lsas) {
    BufferedImage flr = new BufferedImage(img.getWidth(), img.getHeight(), img.getType());
    BufferedImage brzeg = new BufferedImage(img.getWidth()+2*(lsas/2),
img.getHeight()+2*(lsas/2),img.getType());
    int[][] tab = new int[3][lsas*lsas];
    lsas=lsas/2;
    for(int i=0;i<lsas;i++){
        for(int j=0;j<lsas;j++){
            brzeg.setRGB(brzeg.getWidth()-1-i,j,img.getRGB(img.getWidth()-1,0));
            brzeg.setRGB(i, brzeg.getHeight()-1-j, img.getRGB(0, img.getHeight()-1));
            brzeg.setRGB(i,j, img.getRGB(0, 0));
            brzeg.setRGB(brzeg.getWidth()-1-i, brzeg.getHeight()-1-
j,img.getRGB(img.getWidth()-1, img.getHeight()-1));
        }

        for(int xx=0;xx<img.getWidth();xx++){
```

```
                brzeg.setRGB(xx+1+lsas, i, img.getRGB(xx, 0));
                brzeg.setRGB(xx+1+lsas, brzeg.getHeight()-1-i, img.getRGB(xx, img.getHeight()-
1));
            }
            for(int yy=0;yy<img.getHeight();yy++){
                brzeg.setRGB(i, yy+lsas, img.getRGB(0,yy));
                brzeg.setRGB(brzeg.getWidth()-1-i, yy+lsas, img.getRGB(img.getWidth()-1, yy));
            }
        }

        for(int xxx=0;xxx<img.getWidth();xxx++)
            for(int yyy=0;yyy<img.getHeight();yyy++)
                brzeg.setRGB(xxx+lsas, yyy+lsas, img.getRGB(xxx, yyy));

        for (int i=0; i<img.getWidth();i++){
            for(int j=0; j<img.getHeight();j++){
                int gg=0;
                for(int k=-lsas;k<=lsas;k++)
                    for(int g=-lsas;g<=lsas;g++){
                        tab[0][gg]=getR(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        tab[1][gg]=getG(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        tab[2][gg]=getB(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        gg++;
                    }
                Arrays.sort(tab[0]);
                Arrays.sort(tab[1]);
                Arrays.sort(tab[2]);
                flr.setRGB(i,                                                                    j,
toRGB(tab[0][tab[0].length/2],tab[1][tab[1].length/2],tab[2][tab[2].length/2]));

            }
        }

        return flr;
}
```

Na podstawie zadanego otoczenia badanego piksela wybierana jest nowa wartość poprzez wybranie środkowej wartości z otoczenia piksela i podmiana nim starej wartości. W wypadku wyjścia poza zakres na brzegu obrazka brane są wartości skrajne.

e) Ulepszony filtr medzianowy.

```
private static BufferedImage filtr_medianowy2(BufferedImage img, int lsas) {
    BufferedImage flr = new BufferedImage(img.getWidth(), img.getHeight(), img.getType());
    BufferedImage brzeg = new BufferedImage(img.getWidth()+2*(lsas/2),
img.getHeight()+2*(lsas/2),img.getType());
    int[][] tab = new int[3][lsas*lsas];
    lsas=lsas/2;
    for(int i=0;i<lsas;i++){
```

```
            for(int j=0;j<lsas;j++){
                brzeg.setRGB(brzeg.getWidth()-1-i,j,img.getRGB(img.getWidth()-1,0));
                brzeg.setRGB(i, brzeg.getHeight()-1-j, img.getRGB(0, img.getHeight()-1));
                brzeg.setRGB(i,j, img.getRGB(0, 0));
                brzeg.setRGB(brzeg.getWidth()-1-i,                    brzeg.getHeight()-1-
j,img.getRGB(img.getWidth()-1, img.getHeight()-1));
            }

            for(int xx=0;xx<img.getWidth();xx++){
                brzeg.setRGB(xx+1+lsas, i, img.getRGB(xx, 0));
                brzeg.setRGB(xx+1+lsas, brzeg.getHeight()-1-i, img.getRGB(xx, img.getHeight()-
1));
            }
            for(int yy=0;yy<img.getHeight();yy++){
                brzeg.setRGB(i, yy+lsas, img.getRGB(0,yy));
                brzeg.setRGB(brzeg.getWidth()-1-i, yy+lsas, img.getRGB(img.getWidth()-1, yy));
            }
        }

        for(int xxx=0;xxx<img.getWidth();xxx++)
            for(int yyy=0;yyy<img.getHeight();yyy++)
                brzeg.setRGB(xxx+lsas, yyy+lsas, img.getRGB(xxx, yyy));

        for (int i=0; i<img.getWidth();i++){
            for(int j=0; j<img.getHeight();j++){
                int gg=0;
                for(int k=-lsas;k<=lsas;k++)
                    for(int g=-lsas;g<=lsas;g++){
                        tab[0][gg]=getR(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        tab[1][gg]=getG(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        tab[2][gg]=getB(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        gg++;
                    }
                Arrays.sort(tab[0]);
                Arrays.sort(tab[1]);
                Arrays.sort(tab[2]);

                int r,g,b;
                if(0.2*tab[0].length>=1){
                if(Arrays.binarySearch(tab[0],                    getR(img.getRGB(i,
j)))<=0.2*tab[0].length||Arrays.binarySearch(tab[0],                    getR(img.getRGB(i,
j)))>=0.8*tab[0].length)
                    r=tab[0][tab[0].length/2];
                else
                    r=getR(img.getRGB(i, j));
                if(Arrays.binarySearch(tab[1],                    getG(img.getRGB(i,
j)))<=0.2*tab[1].length||Arrays.binarySearch(tab[1],                    getR(img.getRGB(i,
j)))>=0.8*tab[1].length)
                    g=tab[1][tab[1].length/2];
```

```
                else
                        g=getG(img.getRGB(i, j));
                if(Arrays.binarySearch(tab[2], getR(img.getRGB(i,
j)))<=0.2*tab[2].length||Arrays.binarySearch(tab[2], getR(img.getRGB(i,
j)))>=0.8*tab[2].length)
                        b=tab[2][tab[2].length/2];
                else
                        b=getB(img.getRGB(i, j));
                }
                else {
                        r=tab[0][tab[0].length/2];
                        g=tab[1][tab[1].length/2];
                        b=tab[2][tab[2].length/2];
                }

                flr.setRGB(i, j, toRGB(r,g,b));

            }
        }

        return flr;
}
```

Na podstawie zadanego otoczenia badanego piksela wybierana jest nowa wartość poprzez wybranie środkowej wartości z otoczenia piksela i podmiana nim starej wartości z zastrzeżeniem, że stara wartość musi znajdować się w 20% skrajnych wartości. W wypadku wyjścia poza zakres na brzegu obrazka brane są wartości skrajne.

f) Filtr za pomocą średniej.

```
private static BufferedImage filtr_srednia(BufferedImage img, int lsas) {
        BufferedImage flr = new BufferedImage(img.getWidth(), img.getHeight(), img.getType());
        BufferedImage brzeg = new BufferedImage(img.getWidth()+2*(lsas/2),
img.getHeight()+2*(lsas/2),img.getType());
        int[][] tab = new int[3][lsas*lsas];
        lsas=lsas/2;
        for(int i=0;i<lsas;i++){
            for(int j=0;j<lsas;j++){
                brzeg.setRGB(brzeg.getWidth()-1-i,j,img.getRGB(img.getWidth()-1,0));
                brzeg.setRGB(i, brzeg.getHeight()-1-j, img.getRGB(0, img.getHeight()-1));
                brzeg.setRGB(i,j, img.getRGB(0, 0));
                brzeg.setRGB(brzeg.getWidth()-1-i, brzeg.getHeight()-1-
j,img.getRGB(img.getWidth()-1, img.getHeight()-1));
            }

            for(int xx=0;xx<img.getWidth();xx++){
                brzeg.setRGB(xx+1+lsas, i, img.getRGB(xx, 0));
```

```
                brzeg.setRGB(xx+1+lsas, brzeg.getHeight()-1-i, img.getRGB(xx, img.getHeight()-
1));
            }
            for(int yy=0;yy<img.getHeight();yy++){
                brzeg.setRGB(i, yy+lsas, img.getRGB(0,yy));
                brzeg.setRGB(brzeg.getWidth()-1-i, yy+lsas, img.getRGB(img.getWidth()-1, yy));
            }
        }

        for(int xxx=0;xxx<img.getWidth();xxx++)
            for(int yyy=0;yyy<img.getHeight();yyy++)
                brzeg.setRGB(xxx+lsas, yyy+lsas, img.getRGB(xxx, yyy));

        for (int i=0; i<img.getWidth();i++){
            for(int j=0; j<img.getHeight();j++){
                int gg=0;
                for(int k=-lsas;k<=lsas;k++)
                    for(int g=-lsas;g<=lsas;g++){
                        tab[0][gg]=getR(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        tab[1][gg]=getG(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        tab[2][gg]=getB(brzeg.getRGB(i+k+lsas, j+g+lsas));
                        gg++;
                    }
                int r=0,g=0,b=0;
                for(int z=0;z<tab[0].length;z++)
                    r+=tab[0][z];
                for(int z=0;z<tab[1].length;z++)
                    g+=tab[1][z];
                for(int z=0;z<tab[1].length;z++)
                    b+=tab[2][z];
                r/=tab[0].length;
                g/=tab[1].length;
                b/=tab[2].length;
                flr.setRGB(i, j, toRGB(r,g,b));

            }
        }

        return flr;
}
```

Na podstawie zadanego otoczenia badanego piksela wybierana jest nowa wartość poprzez obliczenie średniej ze wszystkich pikseli znajdujących się w otoczeniu wokół piksela.

g) Histogram

```
private static void histogram(String s, BufferedImage img) throws IOException {
	FileWriter fw = new FileWriter(s);
	int his[][] = new int[4][256];
	BufferedImage szr=szarosc1(img);
	int r,g,b,sz;
	for(int i=0;i<img.getWidth();i++)
		for(int j=0;j<img.getHeight();j++)
		{
			r=getR(img.getRGB(i, j));
			g=getG(img.getRGB(i, j));
			b=getB(img.getRGB(i, j));
			sz=getR(szr.getRGB(i, j));
			his[0][r]++;
			his[1][g]++;
			his[2][b]++;
			his[3][sz]++;
		}
		for(int i=0;i<img.getHeight();i++)
			fw.write(i+" "+his[0][i]+" "+his[1][i]+" "+his[2][i]+"
"+his[3][i]+"\n");
		fw.close();
}
```

Zapisuje do pliku w 5 kolumnach histogram. W pierwszej kolumnie wartość piksela w kolejnych ilość wystąpień danej wartości w kanałach R, G i B oraz w obrazie w skali szarości.

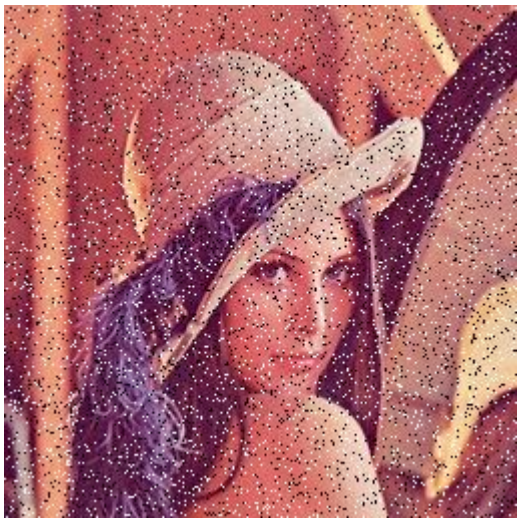
III. **Wyniki i wnioski**

Poniżej prezentuję kolejno otrzymane wyniki działania wyżej przedstawionych funkcji na obrazie *lena.jpg*, który w wersji oryginalnej przedstawiam poniżej.

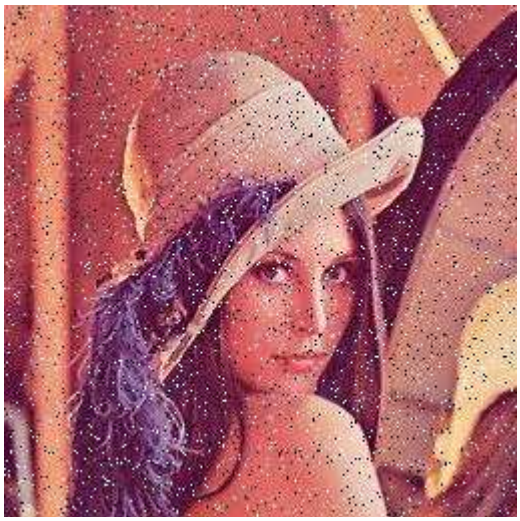
Rys. 1. Oryginalna wersja obrazu *lena.jpg*.

a) Szum typu sól i pieprz.

Rys. 2. Obraz *lena.jpg* z szumem sól i pieprz z p=0.85.



Rys. 3. Obraz *lena.jpg* z szumem sól i pieprz z p=0.90



Rys. 4 Obraz *Lena.jpg* z szumem sól i pieprz.
p=0.95

Na powyższych trzech obrazach widać w jaki sposób wygląda obraz ze sztucznie naniesionym szumem typu sól i pieprz. Taki szum naturalnie może wystąpić w przypadku znacznego uszkodzenia obiektywu.

b) Szum równomierny – zmiana każdego kanału o tę samą wartość.

Rys. 5. Obraz *lena.jpg* z szumem równomiernym z p=0.85.

Rys. 6. Obraz *lena.jpg* z szumem równomiernym z p=0.90.

Rys. 7. Obraz *lena.jpg* z szumem równomiernym z p=0.95

Szum wprowadzony został poprzez zmianę wartości każdego kanału RGB o tę samą wartość z ustalonego przedziału.

c) Szum równomierny2 – zmiana każdego kanału o inną wartość samą wartość.

Rys. 8. Obraz *lena.jpg* z szumem równomiernym drugiego typu z p=0.85.

Rys. 9. Obraz *lena.jpg* z szumem równomiernym drugiego typu z p=0.90.

Rys. 10. Obraz *lena.jpg* z szumem równomiernym drugiego typu z p=0.95

W przeciwieństwie do szumu równomiernego wprowadzanego poprzednim sposobem podanym wyżej, tutaj szum wprowadzany jest poprzez dodanie lub odjecie liczby z zadanego przedziału jednak dla każdego z kanałów RGB losowana jest inna wartość z tego przedziału. Szum taki naturalnie może występować na przykład w wypadku pobrudzenia obiektywu

d) Działanie filtru medianowego.

Powyżej prezentowany jest sposób działania filtru medianowego na obrazach, na których został sztucznie wprowadzony szum typu sól i pieprz. Wykorzystane tutaj było sąsiedztwo 3x3 kolejno dla prawdopodobieństw 0.85, 0.90 i 0.95.

Powyżej zaprezentowany sposób działania filtru medianowego na otoczeniu 3x3 i kolejno prawdopodobieństw 0.85, 0.90, 0.95 dla szumu równomiernego typu jeden.

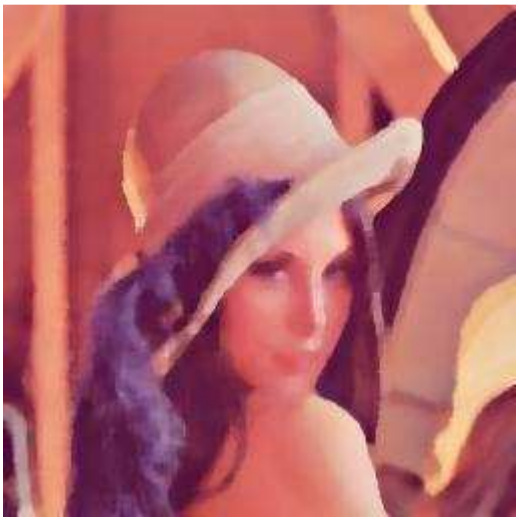
Powyżej zaprezentowany sposób działania filtru medianowego na otoczeniu 3x3 i kolejno prawdopodobieństw 0.85, 0.90, 0.95 dla szumu równomiernego typu dwa.



Działanie filtru medianowego na obrazach ze sztucznie wprowadzonym szumem typu sól i pieprz na otoczeniu wielkości 5x5. Kolejno dla prawdopodobieństw wystąpienia szumu 0.85, 0.90 i 0.95.


Działanie filtru medianowego na obrazach ze sztucznie wprowadzonym szumem równomiernym odpowiednio dla prawdopodobieństw 0.85, 0.90 i 0.95 i otoczeniu 5x5.

Powyżej tak jak poprzednio otoczenie 5x5, kolejno prawdopodobieństwa 0.85, 0.90 i 0.95. Szum równomierny drugiego typu.



Powyżej filtr medianowy zastosowany na obrazach z wprowadzonym szumem typu sól i pieprz kolejno z prawdopodobieństwem 0.85, 0.90 i 0.95. Otoczenie 7x7.


Filtr medianowy na obrazach z wprowadzonym szumem równomiernym pierwszego typu. Kolejno prawdopodobieństwa 0.85, 0.90, 0.95. Otoczenie 7x7.

Powyżej filtr medianowy na obrazach z wprowadzonym szumem równomiernym drugiego typu. Prawdopodobieństwa kolejno 0.85, 0.90 i 0.95. Otoczenie 7x7.

Widać, że po zastosowaniu filtru medianowego uzyskany obraz ulega znacznemu rozmyciu wraz ze wzrostem wielkości otoczenia. Dla otoczenia 7x7 obraz staje się bardzo rozmyty i trudno dostrzec szczegóły. Dla otoczenia 3x3 obraz jest znacznie mniej rozmyty, ale w przypadku dużego zaszumienia nie do końca udało się pozbyć szumów – co najłatwiej zauważyć dla szumu typu sól i pieprz z 15% szumem. Niestety z tak mocno zaszumionym obrazem trudno przeprowadzić operację odszumiania, bo następuje zbyt duża utrata informacji.

e) Działanie ulepszonego filtru medianowego.

Ulepszony filtr medianowy działa podobnie do zwykłego filtru medianowego z zastrzeżeniem, że aby nastąpiła zmiana piksela, to oryginalna wartość piksela musi znajdować się po posortowaniu w 20% skrajnych wartości pikseli z otoczenia.

Powyżej działanie ulepszonego filtru medianowego dla zaszumienia typu sól i pieprz i otoczenia 3x3. Kolejno prawdopodobieństwa wystąpienia szumów 0.85, 0.90 i 0.95.

Powyżej ulepszony filtr medianowy. Szum równomierny typu jeden i otoczenie 3x3.



Ulepszony filtr medianowy. Otoczenie 3x3. Szum równomierny drugiego typu.


Ulepszony filtr medianowy. Otoczenie 5x5. Szum typu sól i pieprz.

Ulepszony filtr medianowy. Otoczenie 5x5. Szum równomierny typu pierwszego.



Ulepszony filtr medianowy. Otoczenie 5x5. Szum równomierny typu drugiego.


Ulepszony filtr medianowy. Otoczenie 7x7. Szum sól i pieprz.

Ulepszony filtr medianowy. Otoczenie 7x7. Szum równomierny pierwszego typu.

Ulepszony filtr medianowy. Otoczenie 7x7. Szum równomierny drugiego typu.

Filtr medianowy sprawdzający dodatkowo czy oryginalna wartość piksela jest jedną z 20% skrajnych wartości w otoczeniu pozwala na uzyskanie obrazu o znacznie lepszej ostrości. Ponieważ poprawiane są tylko te piksele, których wartość odbiega znacznie od wartości pikseli ze swojego otoczenia. Jednak w tym wypadku trzeba wybrać odpowiednio duże otoczenie. Dla otoczenia 3x3 filtr bardzo słabo poradził sobie z wprowadzonymi zakłóceniami – najłatwiej dostrzec to na obrazach z wprowadzonym zakłóceniem typu sól i pieprz. Wprowadzone białe piksele w znacznej części pozostały na obrazie. Nawet dla otoczenia 5x5 jeżeli zakłócenie jest odpowiednio mocne to filtr nie jest w stanie poradzić sobie z całym zakłóceniem. Dopiero dla otoczenia 7x7 widać, że wszystkie zakłócenia zostają usunięte, ale im większe otoczenie tym również większe rozmycie. Jednak różnica jest diametralna w porównaniu do rozmycia powstałego wskutek działania zwykłego filtru medianowego.

f) Działanie filtru za pomocą średniej.

Zamiast zastępowania piksela wartością pikseli z jego otoczenia można nową wartość zastąpić średnią wartości pikseli z otoczenia. W ten sposób działa filtr, którego wyniki przedstawię poniżej.



Filtr za pomocą średniej. Otoczenie 3x3. Szum typu sól i pieprz.


Filtr za pomocą średniej. Otoczenie 3x3. Szum równomierny pierwszego typu.

Filtr za pomocą średniej. Otoczenie 3x3. Szum równomierny drugiego typu.



Filtr za pomocą średniej. Otoczenie 5x5. Szum typu sól i pieprz.

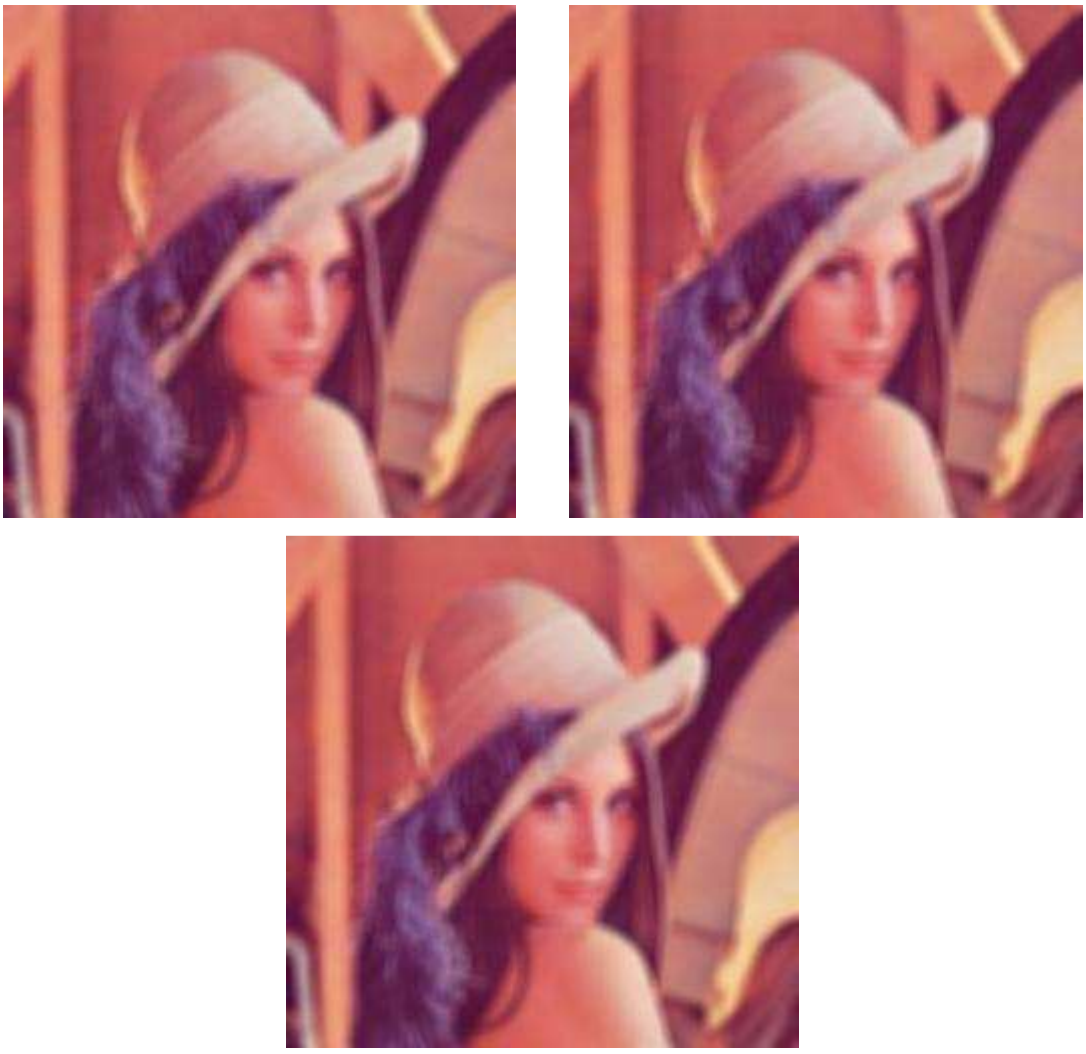

Filtr za pomocą średniej. Otoczenie 5x5. Szum równomierny pierwszego typu.

Filtr za pomocą średniej. Otoczenie 5x5. Szum równomierny drugiego typu.



Filtr za pomocą średniej. Otoczenie 7x7. Szum typu sól i pieprz.


Filtr za pomocą średniej. Otoczenie 7x7. Szum równomierny pierwszego typu.



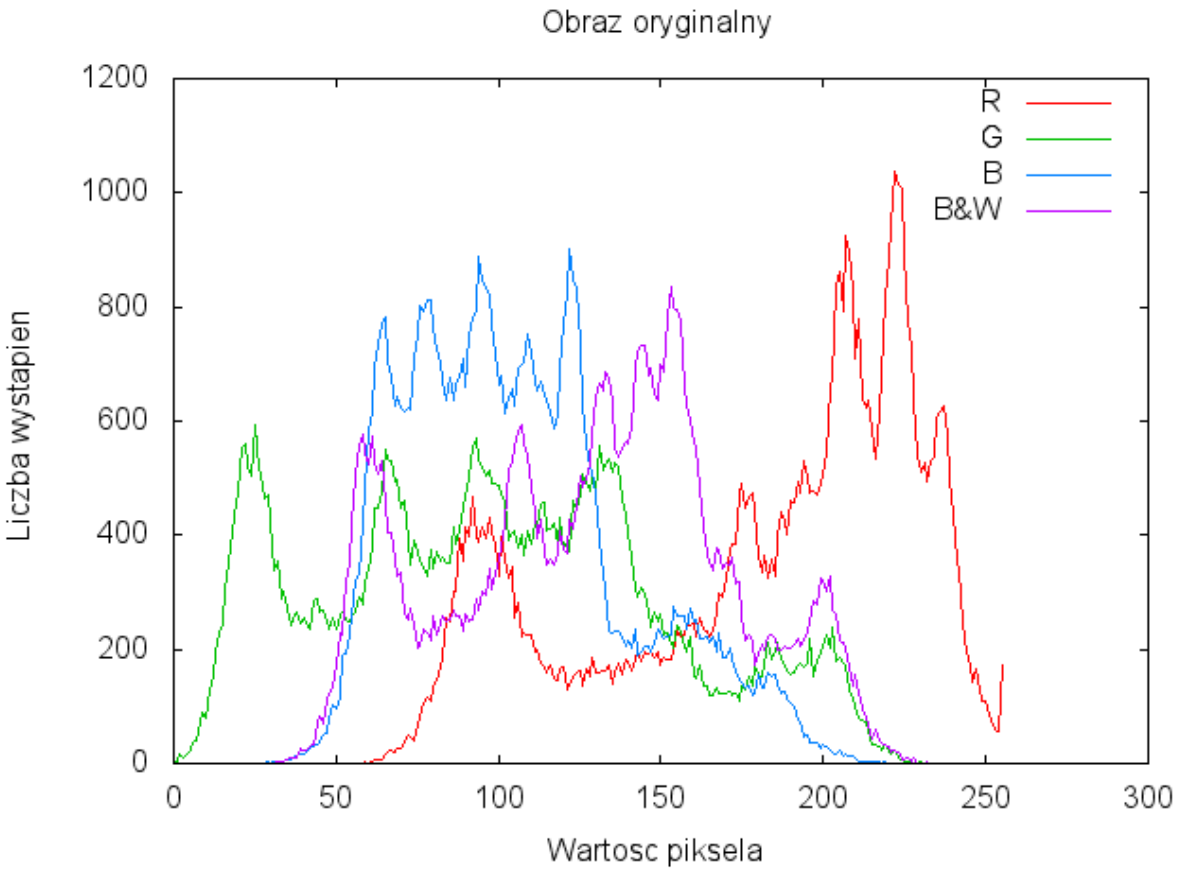
Filtr za pomocą średniej. Otoczenie 7x7. Szum równomierny drugiego typu.

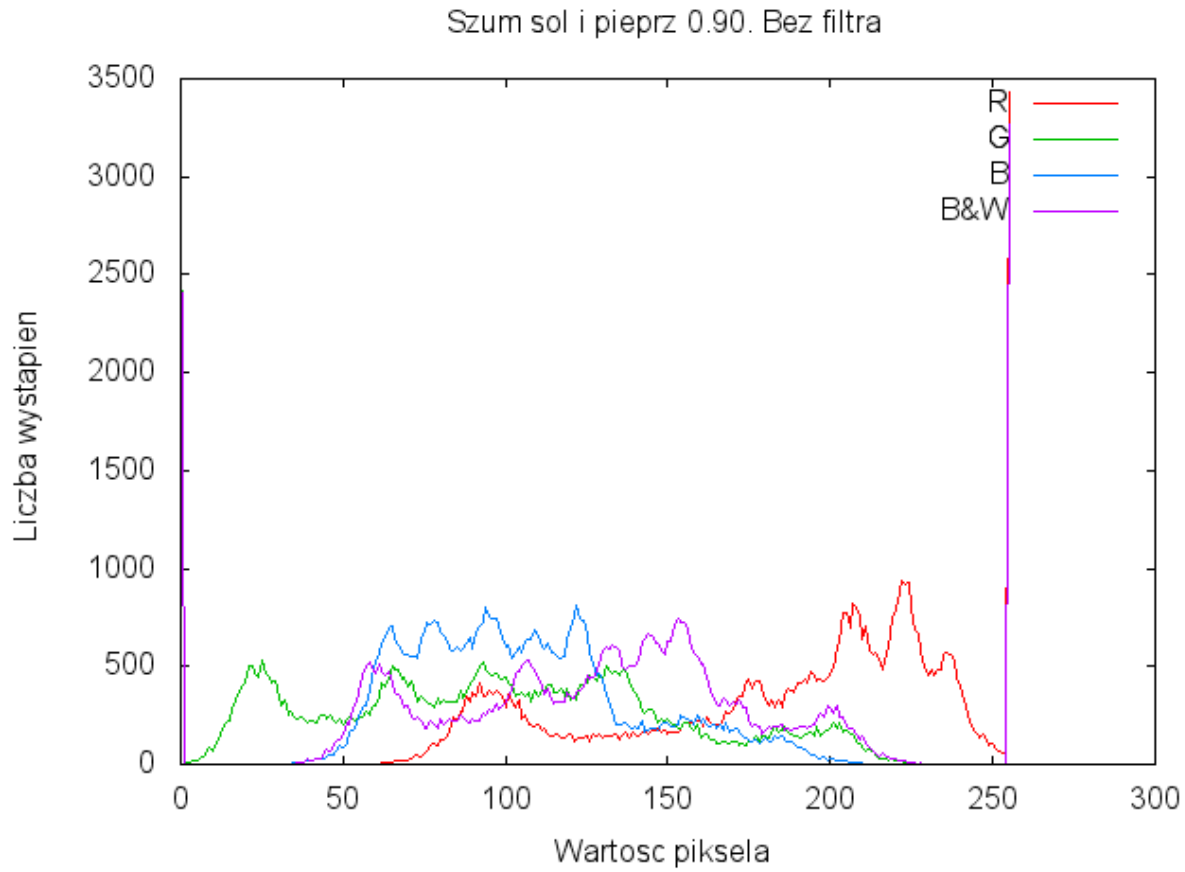
Wykorzystując filtr za pomocą średniej uzyskane rezultaty nie są zadowalające. Obraz w przypadku użycia dużego otoczenia staje się bardzo rozmyty. W przypadku użycia małego otoczenia szum słabo jest usuwany. Jest to najprostszy z wykorzystanych algorytmów, ale jego wynik działania pozostawia dużo do życzenia. W przypadku wystąpienia zakłócenia typu sól i pieprz efekt filtracji wydaje się być gorszy niż przed filtracją dla dużej wartości prawdopodobieństwa wystąpienia zakłócenia.

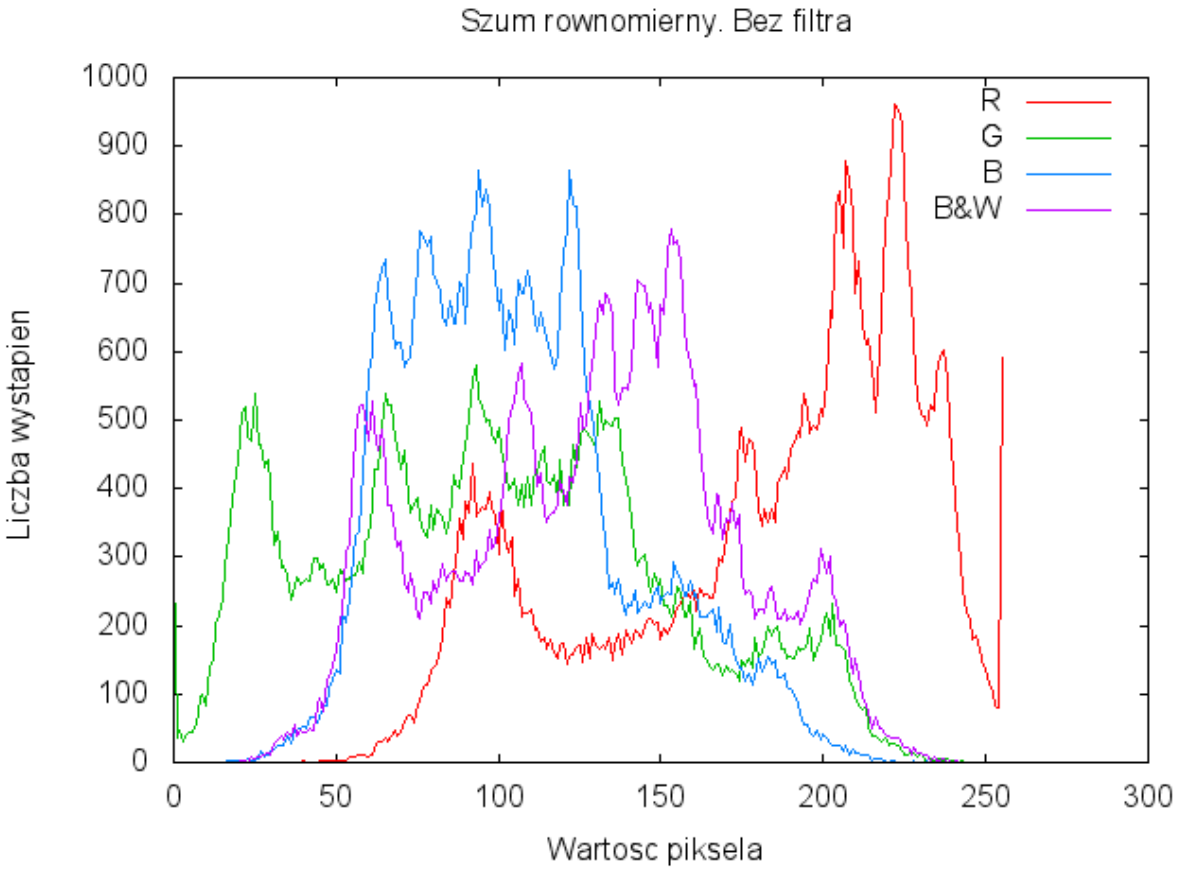
Porównując wyżej zaprezentowane wyniki trzech sposobów filtracji, zdecydowanym faworytem jest ulepszony filtr medianowy. Pozwala na użycie dość dużego otoczenia w taki sposób, że rozmycie jest najmniejsze – nawet porównując wynik działania ulepszonego filtru medianowego na otoczeniu 7x7 z wynikiem działania zwykłego filtru medianowego na otoczeniu 3x3. Bez wątpienia otrzymane wyniki wskazują na ulepszony filtr medianowy jako na najlepszy z użytych podczas laboratoriów filtrów. Natomiast najgorzej w zestawieniu wypada filtr korzystający ze średniej występujących w otoczeniu pikseli. Wyniki pokazują, że nie dość, że słabo daje sobie radę z szumami to w dodatku uzyskuje się korzystając z niego największe rozmycie obrazu. Zwykły filtr medianowy radzi sobie nieco lepiej na małym otoczeniu niż ulepszony filtr medianowy jeżeli ma do czynienia z szumem typu sól i pieprz jednak mimo to robi ogólne wrażenie gorsze niż jego ulepszona wersja.

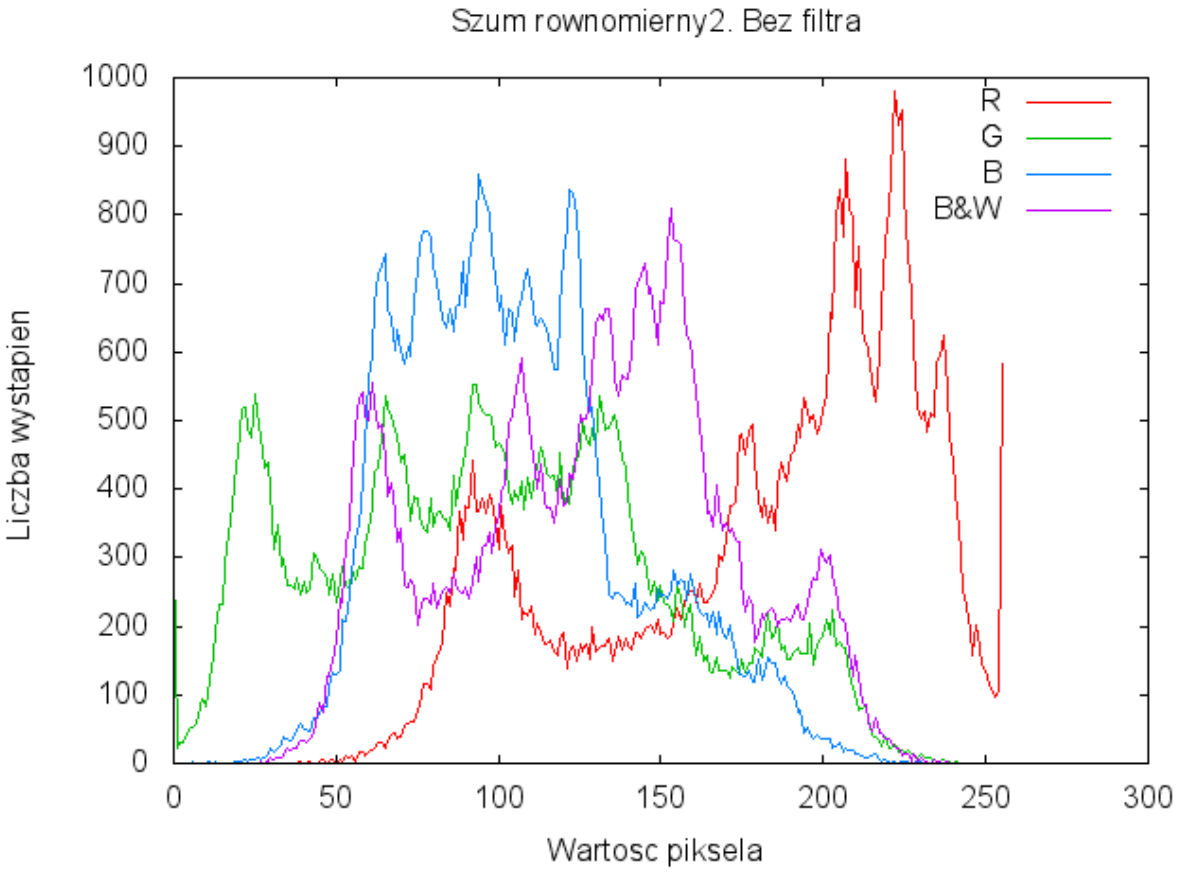
g) Histogramy

Poniżej przedstawiam histogramy dla obrazu oryginalnego, obrazów z wprowadzonym szumem i obrazów po filtracji. Każdy kanał osobno plus skala szarości.



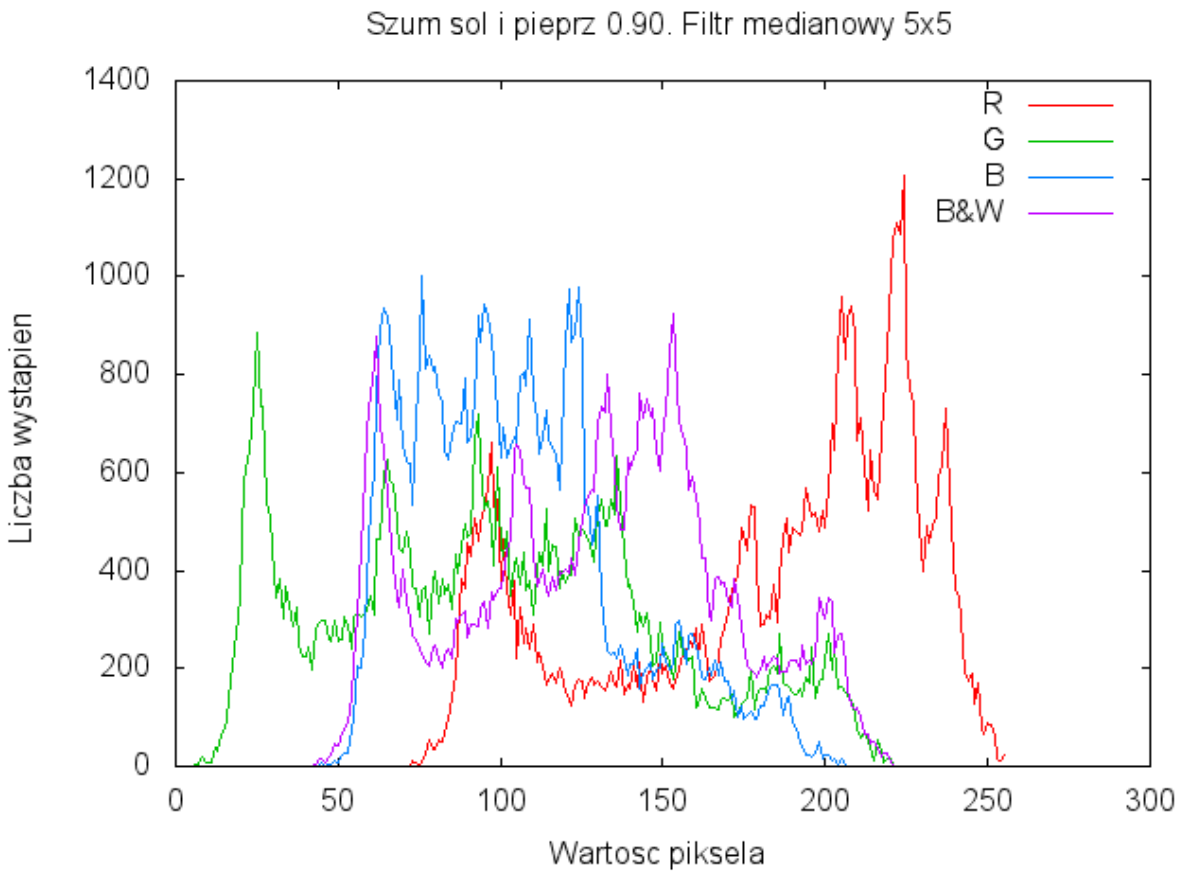


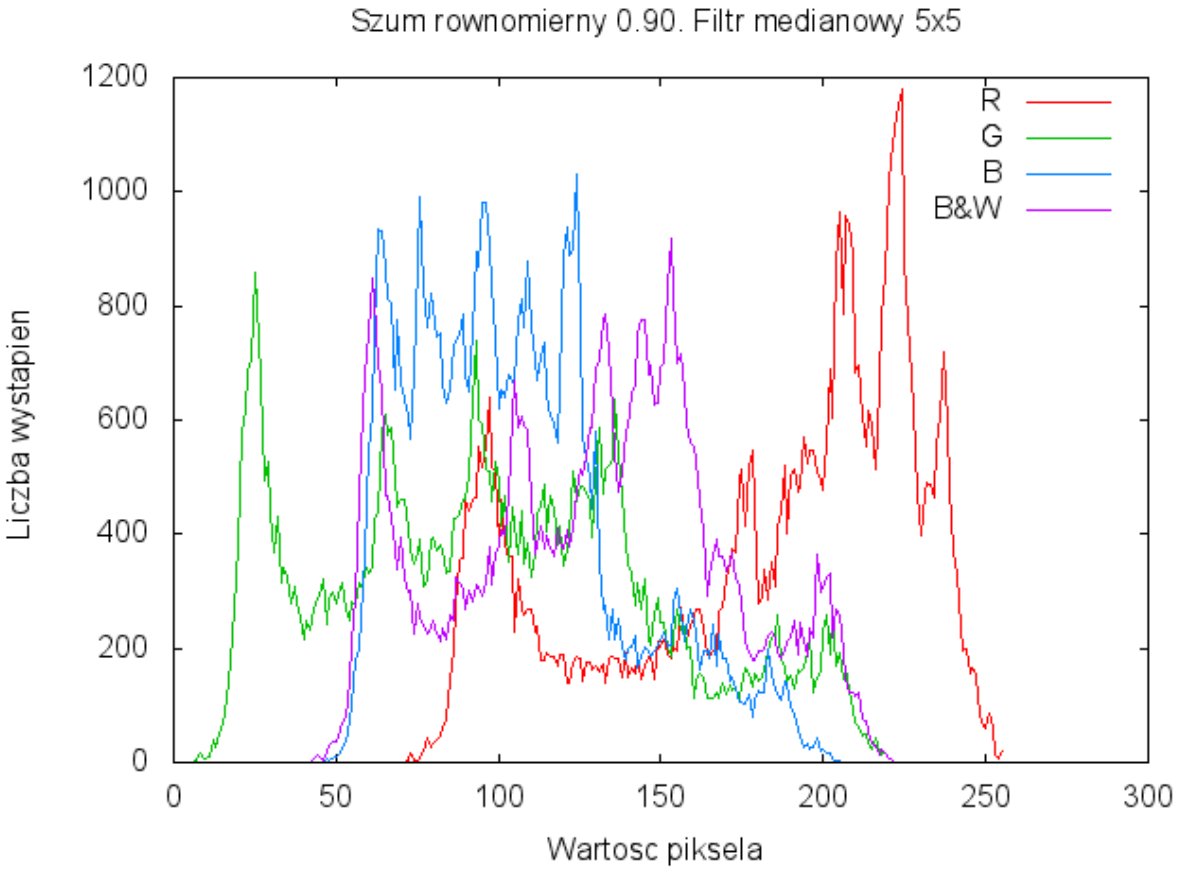

Szum rownomierny. Bez filtra
Liczba wystapien
Wartosc piksela
R
G
B
B&W
0
100
200
300
400
500
600
700
800
900
1000
0
50
100
150
200
250
300

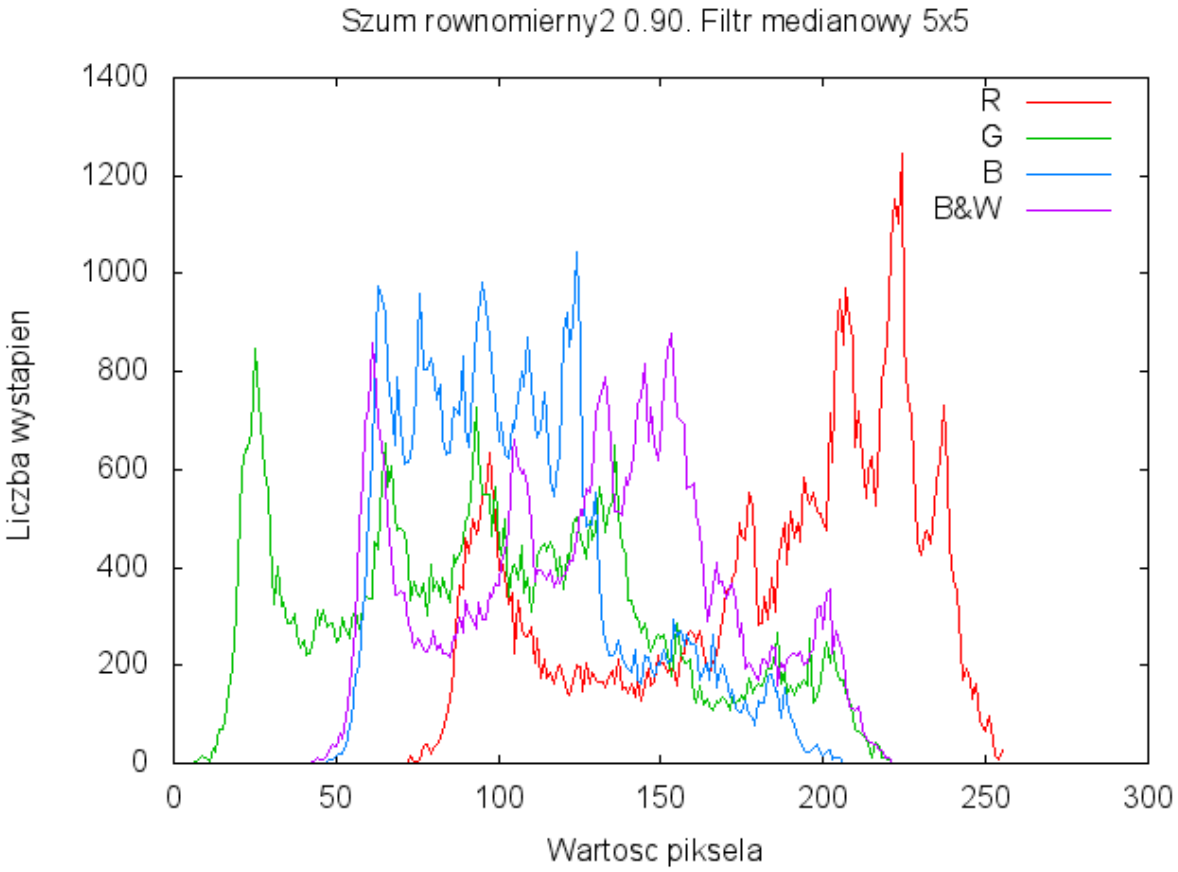

Szum rownomierny2. Bez filtra
Liczba wystapien
Wartosc piksela
R
G
B
B&W
0
100
200
300
400
500
600
700
800
900
1000
0
50
100
150
200
250
300

W porównaniu do histogramu obrazu oryginalnego, gdzie widać duże zróżnicowanie dla każdego kanału RGB po wprowadzeniu szumu równomiernego pierwszego i drugiego typu widać, że histogramy nie ulegają dużej zmiany, nic w tym dziwnego, bo wprowadzane zakłócenia nie były na tyle duże na każdym z kanałów, aby całkowicie zmienić przebieg krzywej. Przy zakłóceniu typu sól i pieprz widać drastyczny wzrost pikseli (0, 0, 0) i (255, 255, 255). Histogramy robione były dla 10% prawdopodobieństwa wprowadzenia zakłóceń. A dla w histogramach poniżej wykorzystane zostało otoczenie 5x5 podczas filtrowania obrazów.

Najpierw przedstawiam histogramy powstałe podczas wykorzystania filtru medianowego kolejno dla zakłóceń sól i pieprz, szumu równomiernego typu jeden i szumu równomiernego typu 2.



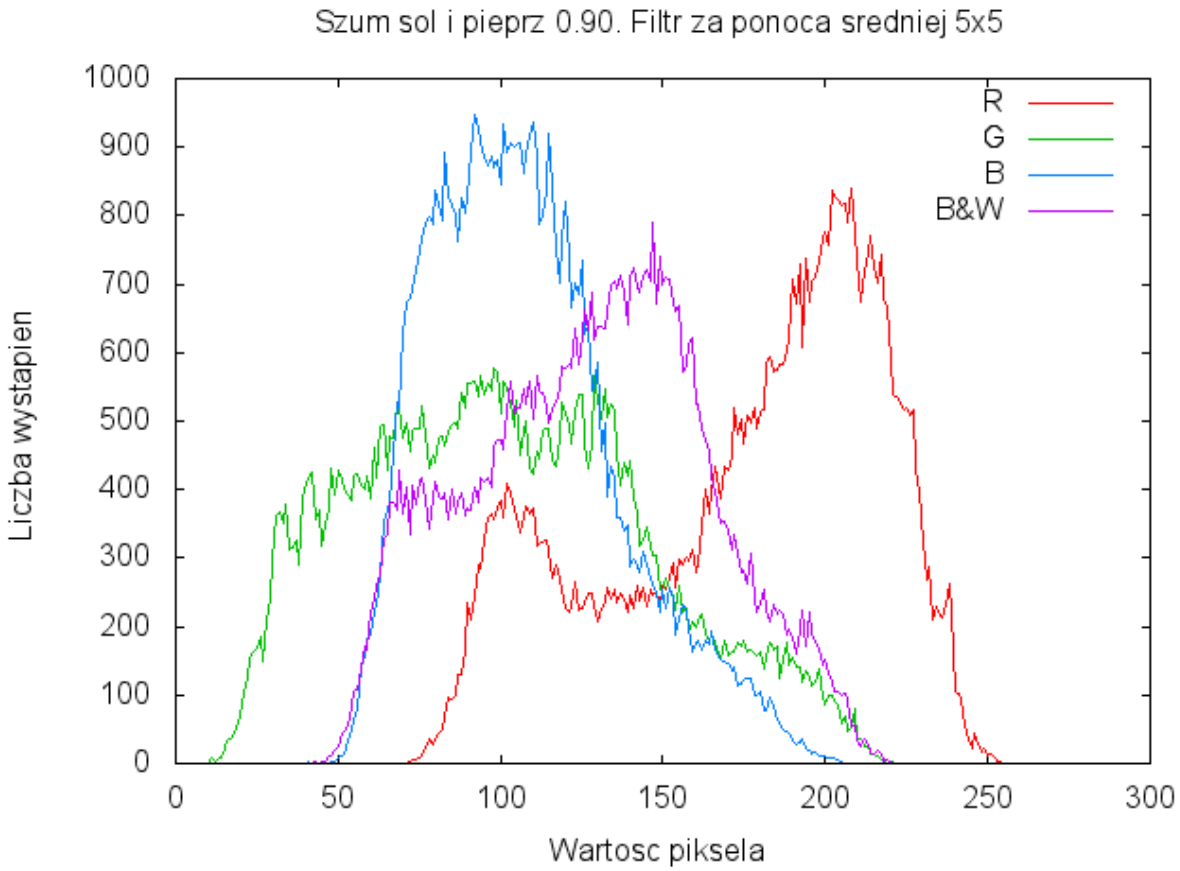

Szum rownomierny 0.90. Filtr medianowy 5x5
Liczba wystapien
Wartosc piksela
R
G
B
B&W
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300

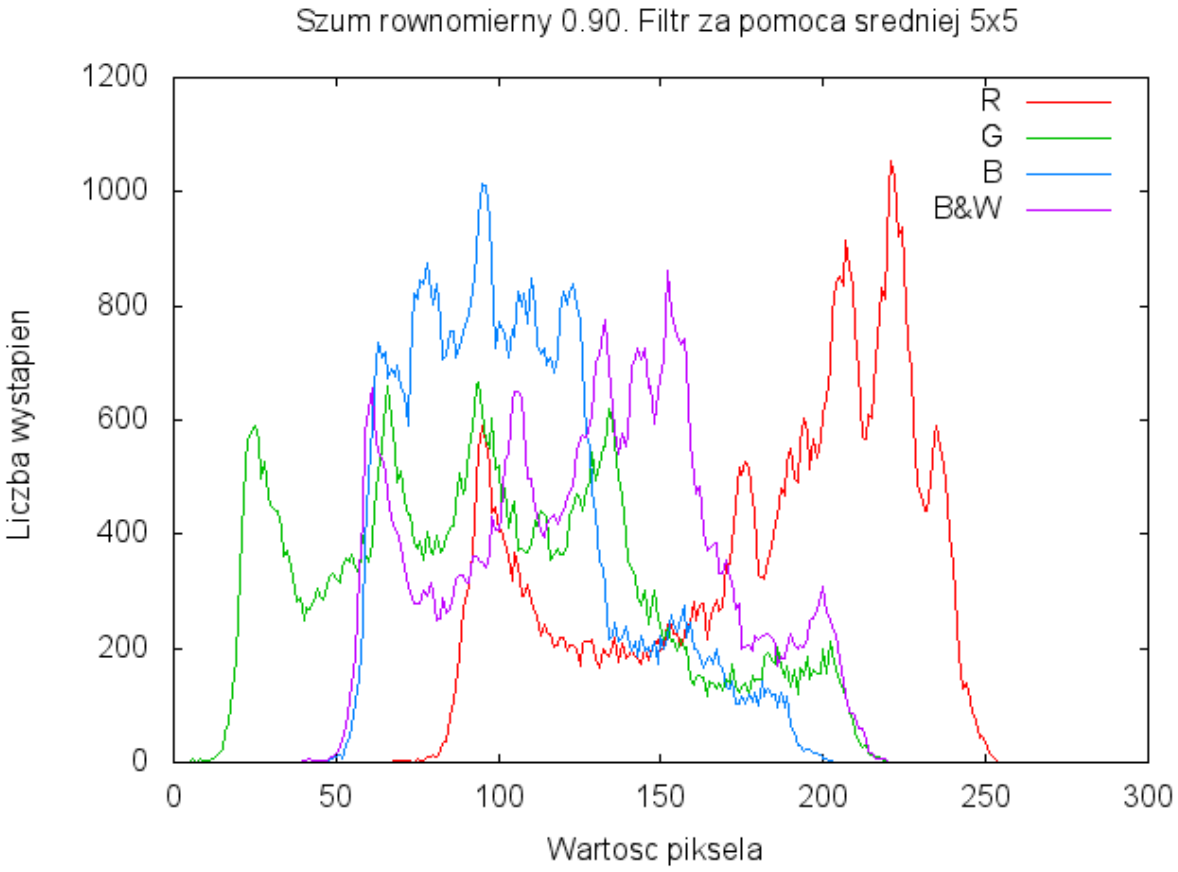

Szum rownomierny2 0.90. Filtr medianowy 5x5
Liczba wystapien
Wartosc piksela
R
G
B
B&W
0
200
400
600
800
1000
1200
1400
0
50
100
150
200
250
300

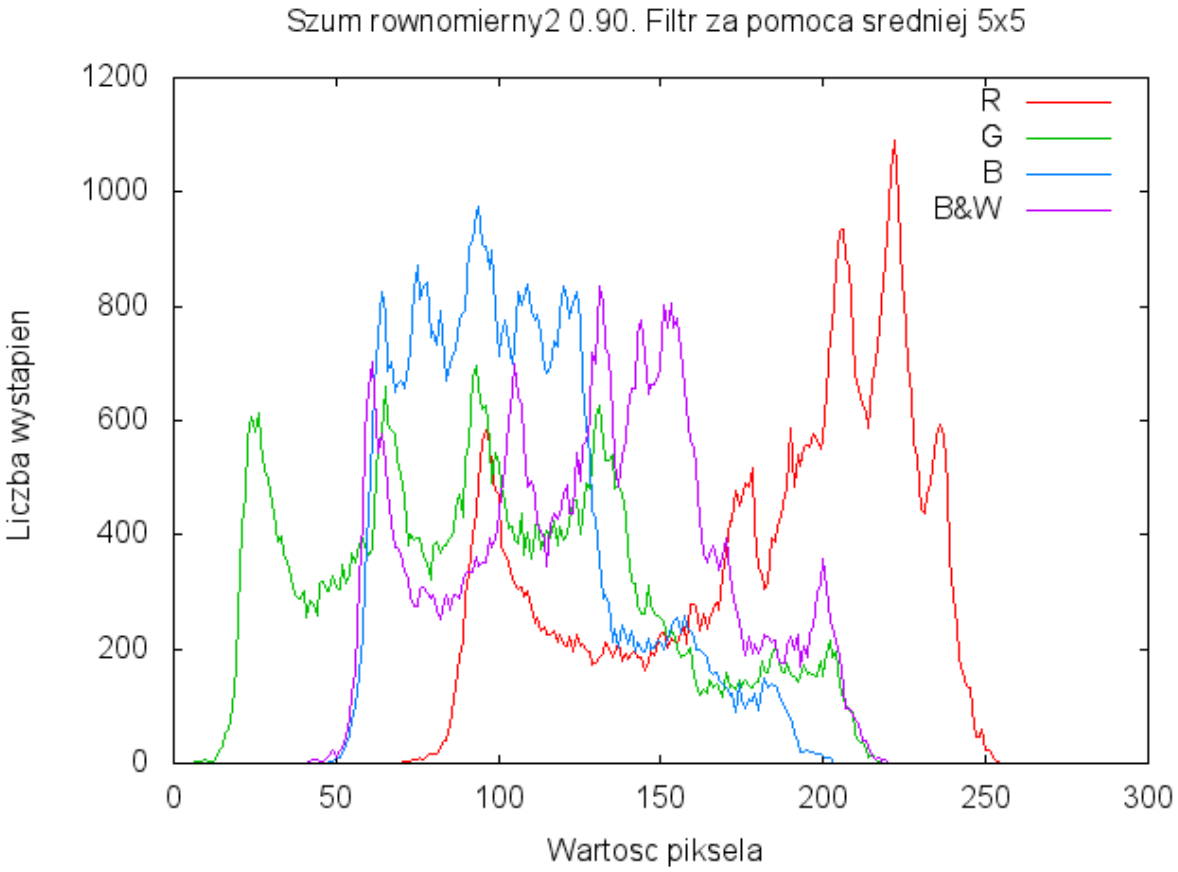
Po wprowadzeniu szumów zauważyć można lekkie przesunięcia w pionie wykresów (nie licząc szumu typu sól i pieprz, który po prostu zmniejsza liczbę wystąpień danej wartości na koszt pikseli czarnych i białych), jednak kształt wykresu praktycznie nie ulega zmianie. Na histogramach po przeprowadzeniu filtracji medianowej widać już znaczne zmiany kształtu. Piksele najbardziej wyróżniające się zostały odfiltrowane a na ich miejsce zostały wprowadzone najmniej skrajne wartości kolorów.

W wypadku histogramów dla ulepszonego filtru medianowego uzyskane histogramy nie będą się znacznie różnić w porównaniu do tych, gdzie wykorzystany był zwykły filtr medianowy z uwagi na to, że sposób działania jest bardzo podobny tylko, że mniej pikseli zostaje zmienionych ze względu na dodatkowy warunek zmiany. Zmiany będą zatem mniejsze, ale ich kształt będzie podobny.

Poniżej zaprezentuję jeszcze histogramy po filtracji filtrem za pomocą średniej:




Szum rownomierny 0.90. Filtr za pomoca sredniej 5x5
Liczba wystapien
Wartosc piksela
R
G
B
B&W
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300


Szum rownomierny2 0.90. Filtr za pomoca sredniej 5x5
Liczba wystapien
Wartosc piksela
R
G
B
B&W
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300

Widać, że po zastosowaniu filtru za pomocą średniej w znacznym stopniu wartości pikseli zostały uśrednione. Objawia się to w postaci ścieśnienia wykresów przedstawiających histogramy. Nowe wartości obliczane są za pomocą średniej, więc skutkuje to tym, że wartości skrajne zostają zastąpione średnią wartości z otoczenia pikseli.

Podsumowując to co zostało zawarte już wcześniej. Najlepsze efekty daje filtracja ulepszonym filtrem medianowym. Dodatkowy warunek skutkujący tym, że tylko skrajne wartości są wymieniane na inne z otoczenia piksela skutkuje tym, że mniej wartości zostaje zastąpionych niepotrzebnie. Rozmycie powstałe po filtracji jest dzięki temu znacznie mniejsze a efekty znacznie lepsze. Najgorszy z filtrów to filtr, który nową wartość piksela liczy jako średnią wartość pikseli ze swojego otoczenia. Obraz zostaje rozmyty, a jeżeli piksele w znacznym stopniu odbiegały od swojej poprawnej wartości – jak przy szumie typu sól i pieprz, to zamiast poprawy jakości zdjęcia uzyskiwaliśmy jej pogorszenie. Zwykły filtr medianowy radził sobie to najlepiej z zaszumieniem typu sól i pieprz na najmniejszym otoczeniu 3x3. Rozmycie nie było takie duże, a szum był dobrze usuwany. Mimo to i tak faworytem tego zestawienia jest ulepszony filtr medianowy, który radził sobie najlepiej z pozostałymi rodzajami szumów.