Programowanie urządzeń mobilnych pod kątem zastosowań biometrycznych

Sprawozdanie I

Temat: Aplikacja wykonująca operacje na obrazach

Autor:
Magda Nowak-Trzos

Prowadzący ćwiczenia:
mgr Krzysztof Misztal

Kraków, 12 kwietnia 2014

Spis treści

1 Temat zadania

Tematem zadania jest napisanie aplikacji na platformę Android wykonującej niżej wymienione operacje na obrazach wraz z któtkim opisem każdej z nich:

- konwersja obrazu na negatyw
- konwersja obrazu na odcienie szarości
- konwersja obrazu na sepię ze współczynnikiem W=20 oraz W=40
- detekcja krawędzi za pomocą krzyża Robertsa
- detekcja krawędzi za pomocą operatora Sobela
- normalizacja histogramu dla obrazu kolrowego i obrazu w odcieniach szarości
- wykrywanie skóry metodą 1 - wykrywanie skóry metodą 2

2 Proponowane rozwiązanie

Aplikacja z użyciem Action Bar'a, umozliwiająca użytkownikowi wybór konktretnej operacji wykonywanej na obrazie. Z menu można również wczytać konkretny obrazek z bazy aplikacji i wykonać na nim wybraną operację. W głownym oknie aplikacji, w centrum znajduje się przerabiany obraz, poniżej 2 przyciski. Jeden umożliwiający przywrócenie oryginału, drugi wyświetlający krótki opis każdej z operacji. Aplikacja umożliwia także wczytanie zdjęcia z galerii - po kliknięciu w obraz - oraz wykonanie na nim każdej z operacji. Ponadto aplikacja została przystosowana do pracy na różnych urządzeniach, oraz w dwóch wersjach językowych - polskiej i angielskiej.

2.1 Krok 1: Szkielet aplikacji

Na początek wykonałam szkielet aplikacji. Umieściłam przykładowy obrazek w środku okna aplikacji oraz stworzyłam Action Bar - menu aplikacji. Każda z operacji wykonuje się po kliknięciu w odpowieni przycisk menu. Ponadto dla takich opcji jak Sepia, wyrównanie histogramu czy wczytanie nowego obrazka uruchamia się dodatkowe pod- menu wyboru dotyczące dokładnego wyboru rodzaju operacji.

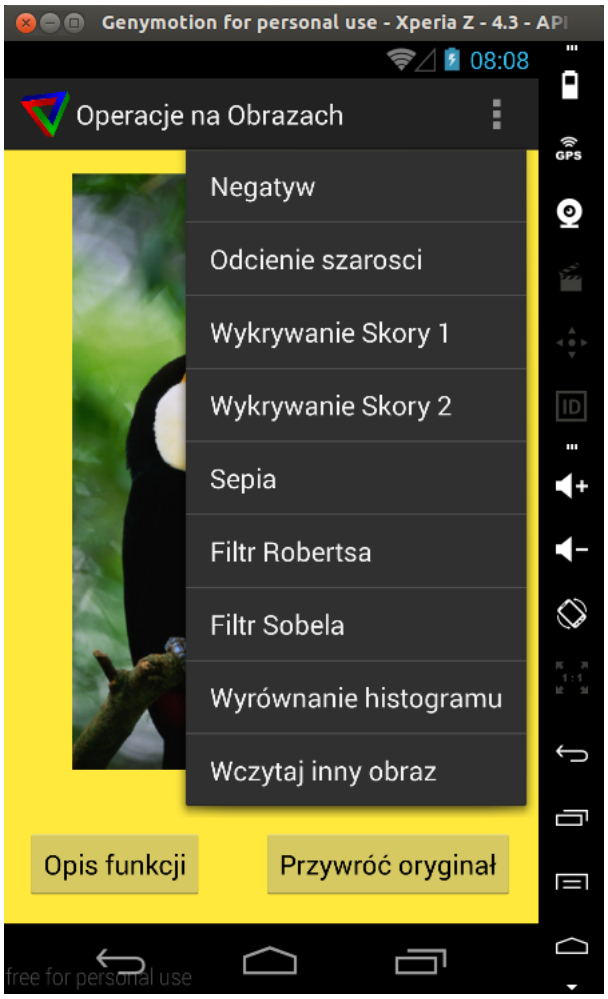
2.1.1 Czynności

Kod menu:

```
1  public boolean onOptionsItemSelected(MenuItem item) {
2          ImageView iv = (ImageView) findViewById(R.id.imageView); // wczytanie obrazu
3          Bitmap sourceBitmap = ((BitmapDrawable) iv.getDrawable()).getBitmap(); // zamiana
               obrazu na bitmape
4          switch(item.getItemId())
5          {
6
7              case R.id.negatyw: // wykonanie operacji negatyw
8                  iv.setImageBitmap(Negatyw(sourceBitmap));
9                  break;
10             case R.id.grey:
11                 iv.setImageBitmap(Grey(sourceBitmap));// Odcienie szarosci
12                 break;
13             case R.id.sepia20:
14                 iv.setImageBitmap(Sepia(sourceBitmap,20)); // Sepia z wspolczynnikiem 20
15                 break;
16             case R.id.sepia40:
17                 iv.setImageBitmap(Sepia(sourceBitmap,40)); // Sepia z wspolczynnikiem 40
18                 break;
19             case R.id.roberts:
20                 iv.setImageBitmap(Roberts(sourceBitmap)); // Filtr Robertsa
21                 break;
22             case R.id.sobel:
23                 iv.setImageBitmap(Sobel(sourceBitmap)); // Filtr Sobela
24                 break;
25             case R.id.histokol:
```

```
26                  iv.setImageBitmap(HistoKol(sourceBitmap)); // wyrównanie histogramu dla
                        obrazu kolorowego
27                  break;
28              case R.id.histogrey:
29                  iv.setImageBitmap(HistoGrey(sourceBitmap)); // wyrównanie histogramu dla
                        obrazu w odcieniach szarości
30                  break;
31              case R.id.wykrywanie1:
32                  iv.setImageBitmap(WykrywanieSkory1(sourceBitmap)); // wykrywanie skóry
                        metodą pierwszą
33                  break;
34              case R.id.wykrywanie2:
35                  iv.setImageBitmap(WykrywanieSkory2(sourceBitmap)); // wykrywanie skóry
                        metodą drugą
36                  break;
37              case R.id.lenna1: // wczytanie obrazka Lenna
38                  ImageView iv4 = (ImageView) findViewById(R.id.lennaim);
39                  oryginal = ((BitmapDrawable) iv4.getDrawable()).getBitmap();
40                  Bitmap lenna = ((BitmapDrawable) iv4.getDrawable()).getBitmap();
41                  iv.setImageBitmap(lenna);
42                  break;
43              case R.id.stwor: //wczytanie obrazka Stwór
44                  ImageView iv3 = (ImageView) findViewById(R.id.stworim);
45                  oryginal = ((BitmapDrawable) iv3.getDrawable()).getBitmap();
46                  Bitmap stwor = ((BitmapDrawable) iv3.getDrawable()).getBitmap();
47                  iv.setImageBitmap(stwor);
48                  break;
49              case R.id.kot: //wczytanie obrazka Kot
50                  ImageView iv2 = (ImageView) findViewById(R.id.kotim);
51                  oryginal = ((BitmapDrawable) iv2.getDrawable()).getBitmap();
52                  Bitmap kot = ((BitmapDrawable) iv2.getDrawable()).getBitmap();
53                  iv.setImageBitmap(kot);
54                  break;
55              case R.id.tucan: //wczytanie obrazka Tucan
56                  ImageView iv5 = (ImageView) findViewById(R.id.tucanim);
57                  oryginal = ((BitmapDrawable) iv5.getDrawable()).getBitmap();
58                  Bitmap tucano = ((BitmapDrawable) iv5.getDrawable()).getBitmap();
59                  iv.setImageBitmap(tucano);
60                  break;
61
62              case R.id.kobieta1: // wczytanie obrazka Kobieta
63                  ImageView iv6 = (ImageView) findViewById(R.id.kobietaim);
64                  oryginal = ((BitmapDrawable) iv6.getDrawable()).getBitmap();
65                  Bitmap kobieta = ((BitmapDrawable) iv6.getDrawable()).getBitmap();
66                  iv.setImageBitmap(kobieta);
67                  break;
68          }
69
70
71          int id = item.getItemId();
72          return super.onOptionsItemSelected(item);
73      }
```

2.1.2 Rezultat



MENU

2.2 Krok 2: Wczytywanie zdjęcia z galerii

Klikając w obraz otwiera się menu dające możliwość wyboru użytkownikowi skąd wczyta obraz. Po wybraniu galerii i kliknięciu w konkretny obraz, jest on przesyłany do aplikacj i można na nim wykonywać już wszystkie operacje.

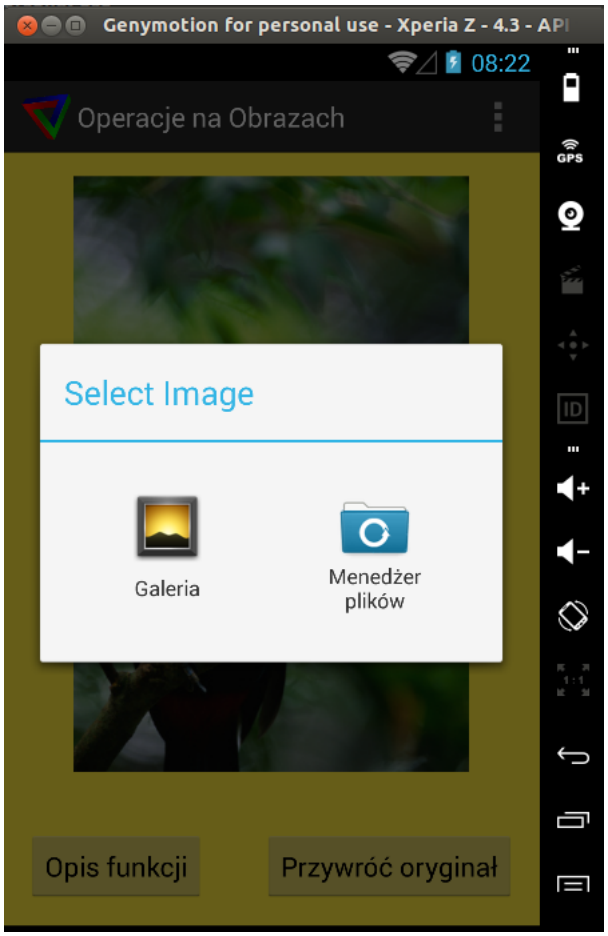
2.2.1 Czynności

Kod odpowiadający z wybór zdjęcia z galerii: (jest on uruchamiany po kliknięciu z obraz)

```
1  public void Galeria(View view) {
2
3          Intent gallery = new Intent(); // nowe okno
4          gallery.setType("image/*"); // jaki typ danych wczytywany
5          gallery.setAction(Intent.ACTION_GET_CONTENT); // ustawienie akcji
6          startActivityForResult(Intent.createChooser(gallery,"Select Image"),1); //
               uruchomienie dodatkowego menu wyboru
7
8      }
9
10     public void onActivityResult(int requestCode , int resultCode, Intent data)
11     {
```

```
12
13          if(resultCode  == RESULT_OK) // jezeli wybrano obrazek
14          {
15              if(requestCode ==1)
16              {
17                  image.setImageURI(data.getData()); // ustawianie obrazka
18                  obraz = ((BitmapDrawable) image.getDrawable()).getBitmap();
19                  oryginal= ((BitmapDrawable) image.getDrawable()).getBitmap(); // ustawienie
                        oryginalu
20                  ImageView iv = (ImageView) findViewById(R.id.imageView);
21                  Bitmap sourceBitmap = ((BitmapDrawable) iv.getDrawable()).getBitmap();
22                  iv.setImageBitmap(obraz);
23              }
24          }
25      }
```

2.2.2 Rezultat



Wybór źródła pobrania obrazu

2.3 Krok 3: Opis funkcji

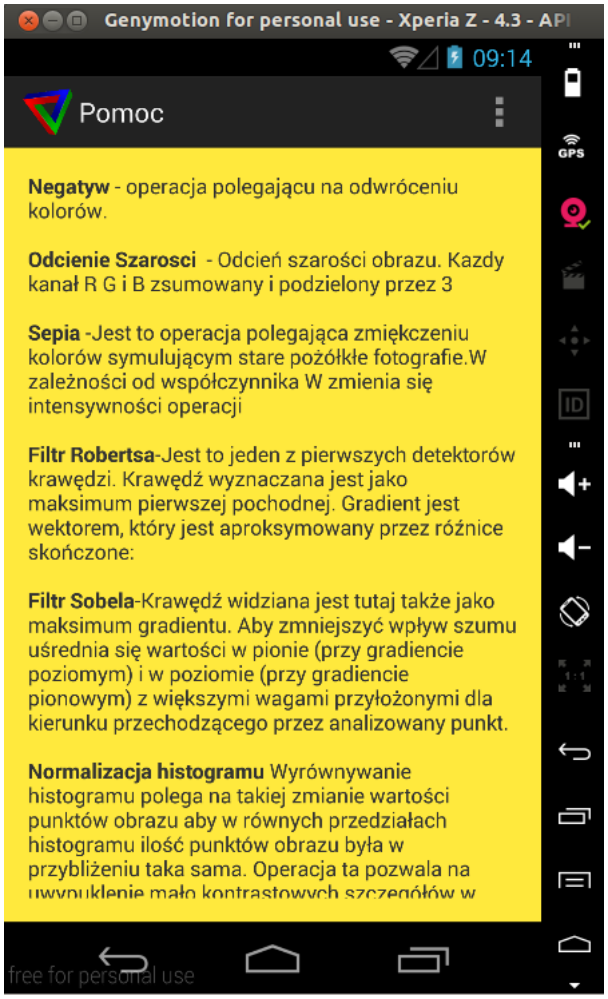
Kolejnym krokiem było stworzenie nowej klasy Help, odpowiedzialnej za nowe okno, w którym umieściłam opis wszystkich używanych funkcji. Pondato, ponieważ tekst był zbyt długi dodałam ScrollView tak aby można go było przewijać.

2.3.1 Czynności

Kod odpowiedzialny za wyswietlenie opisu funkcji

```
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/opis"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:singleLine="false"
        />
    </ScrollView>
```

2.3.2 Rezultat



Zdjęcie oryginalne

2.4 Krok 4: Przywracanie oryginału

Po wykonaniu operacji na obrazie przycisk "Przywracanie oryginału" umożliwia użytkownikowi ponowne wczytanie przerabianego obrazu, ale bez żadnych modyfikacji. Podczas wczytywania obrazu do aplikacji, w polu

```
Bitmap oryginal;
```

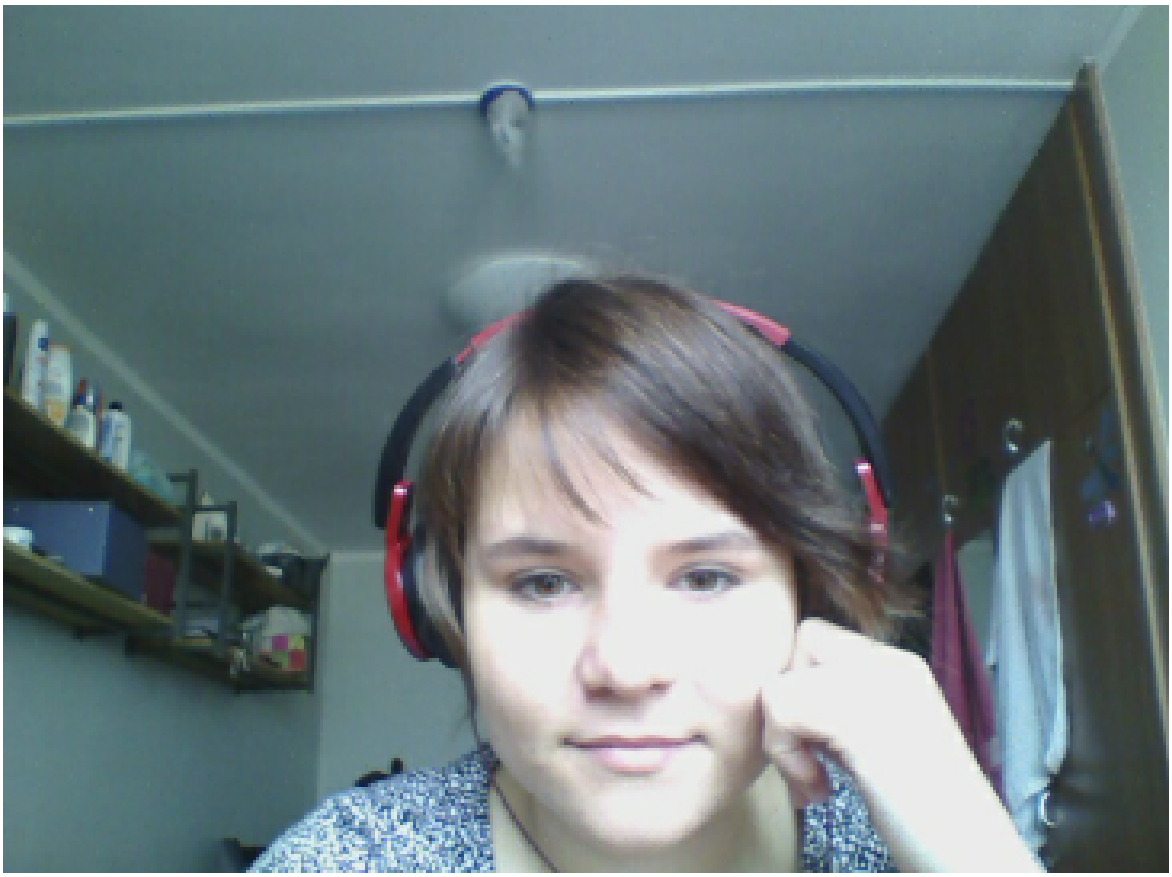
zapisywany jest wczytany obraz. Po kliknięciu w przycisk przywracania oryginału ustawiany jest właśnie ten obraz który wcześniej został zapisany jako oryginał.

2.4.1 Czynności

Kod odpowiedzialny za ustawienie oryginału:

```
public void Oryginal(View v) {

        ImageView iv = (ImageView) findViewById(R.id.imageView);
        Bitmap sourceBitmap = ((BitmapDrawable) iv.getDrawable()).getBitmap();
        iv.setImageBitmap(oryginal);
}
```

2.4.2 Oryginał



Zdjęcie oryginalne

2.5 Krok 5: Konwersja na negatyw

Konwersja na negatyw polega na odwróceniu wartości każdego kanału R G i B dla poszczególnych pikseli. W mojej metodzie tworzę nowy obraz wyjściowy o takich samych rozmiarach jak obraz wejściowy i zapisuje dla każdego z jego pikeli odpowiednią wartośc korzystając z wartości obrazka wejściowego.

2.5.1 Czynności

Kod metody realizującej konwersję na negatyw

```
1   public Bitmap Negatyw(Bitmap src) {
2       Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
            ;
3       int A, R, G, B;
4       int pixelColor;
5       int height = src.getHeight();
6       int width = src.getWidth();
7       for (int y = 0; y < height; y++) {
8           for (int x = 0; x < width; x++) {
9               pixelColor = src.getPixel(x, y);
10              A = Color.alpha(pixelColor);
11              R = 255 - Color.red(pixelColor);
12              G = 255 - Color.green(pixelColor);
13              B = 255 - Color.blue(pixelColor);
14              bmOut.setPixel(x, y, Color.argb(A, R, G, B));
15          }
16      }
17      return bmOut;
18  }
```

2.5.2 Rezultat

Obrazek po zastosowaniu konwersji na negatyw

2.6 Krok 6: Konwersja obrazu na odcienie szarości

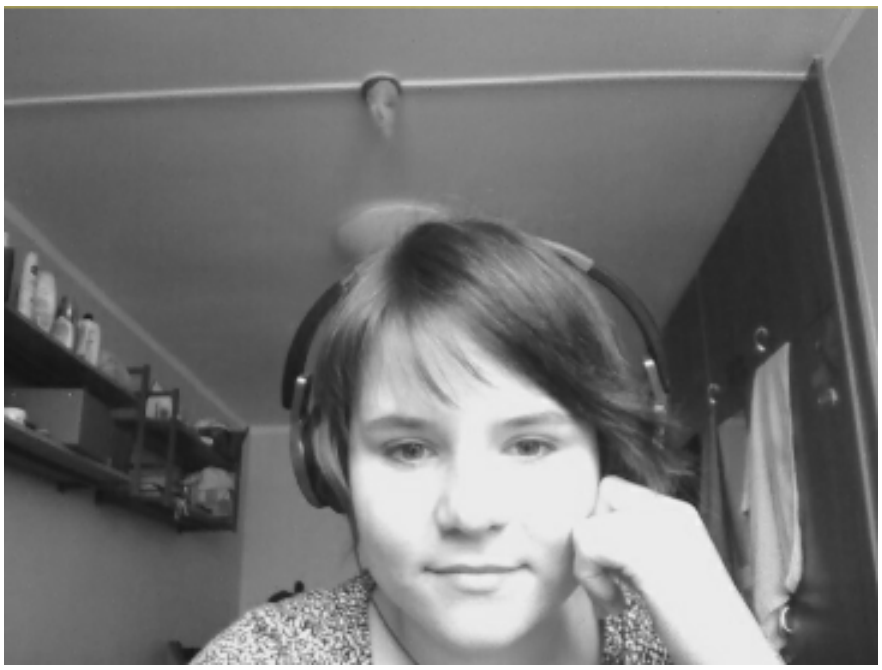
Konwersja obrazu na odcienie szarości polega na ustawieniu wartości każdego kanału R G i B na taką samą - równą średniej arytmetycznej kanałów R G i B z obrazu wejściowego

2.6.1 Czynności

Kod metody realizującej konwersję na obrazu na odcienie szarości

```
1     public Bitmap Gray(Bitmap src) {
2         Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
              ;
3         int Z;
4         int pixelColor;
5         int height = src.getHeight();
6         int width = src.getWidth();
7         for (int y = 0; y < height; y++) {
8             for (int x = 0; x < width; x++) {
9                 pixelColor = src.getPixel(x, y);
10                Z = (Color.red(pixelColor) +Color.green(pixelColor)+Color.blue(pixelColor))
                      /3;
11                bmOut.setPixel(x, y, Color.rgb(Z, Z, Z));
12            }
13        }
14        return bmOut;
15    }
```

2.6.2 Rezultat



Konwersja na odcienie szarości

2.7 Krok 7: Konwersja obrazu na sepię ze współczynnikiem W=20 oraz W=40

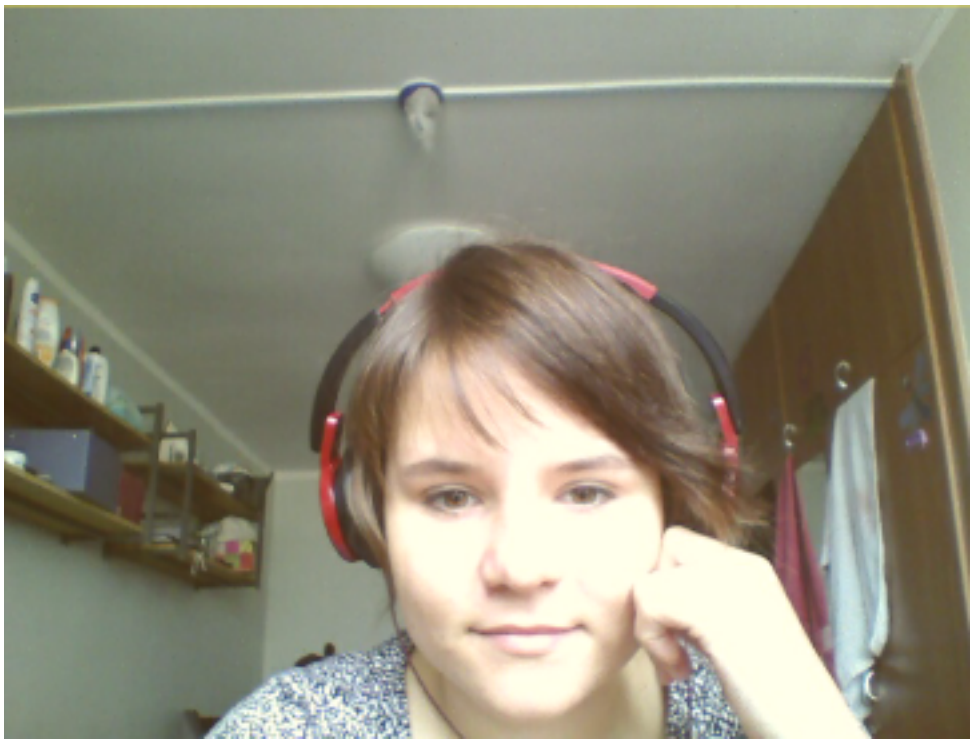
Konwersja obrazu na sepię polega na

2.7.1 Czynności

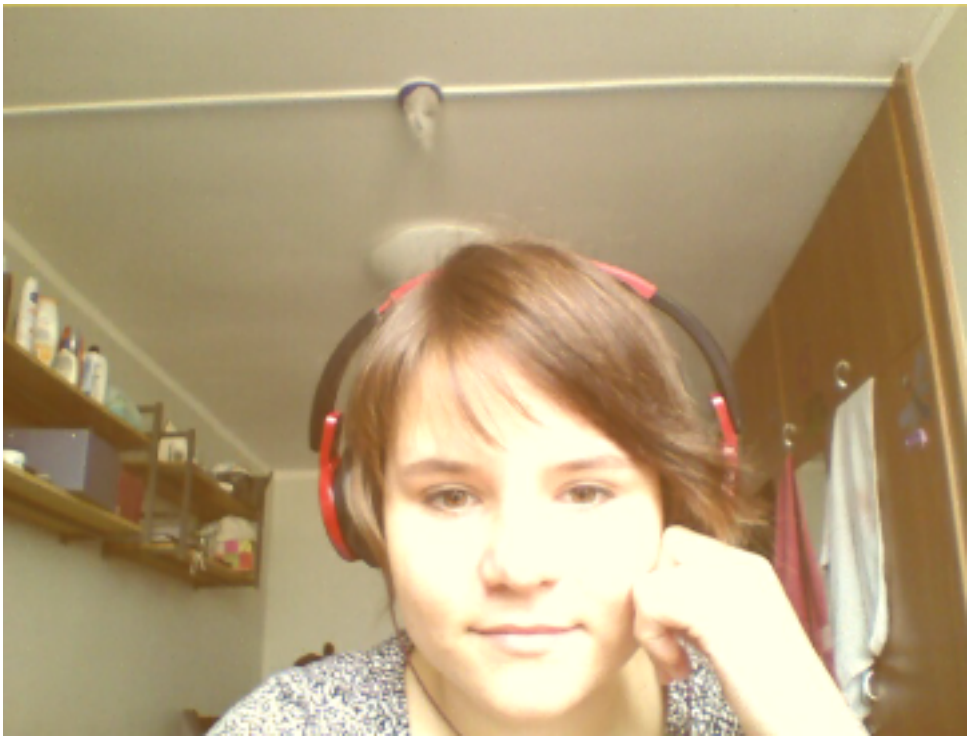
Kod metody realizującej konwersję na sepię z zadanym współczynnikiem polega na dodaniu do kanału czerwonego podwojonej wartości współczynnika, a do kanału zielonego wartości równej wartości współczynnika. W zależności od współczynnika sepia jest słabsza lub mocniejsza. Moja metoda jako argument przyjmuje współczynnik. W aplikacji zastosowałam sepię o wsp = 20 i 40.

```
1
2     public Bitmap Sepia(Bitmap src, int wsp) {
3          Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig()
               );
4         int R, G, B,A;
5         int pixelColor;
6         int height = src.getHeight();
7         int width = src.getWidth();
8         for (int y = 0; y < height; y++) {
9             for (int x = 0; x < width; x++) {
10                 pixelColor = src.getPixel(x, y);
11                A = Color.alpha(pixelColor);
12                R = Color.red(pixelColor)+2*wsp;
13                G = Color.green(pixelColor)+wsp;
14                B = Color.blue(pixelColor);
15                if(R>255)
16                {
17                    R=255;
18                }
19                if(B>255)
20                {
21                    B=255;
22                }
23                if(G>255)
24                {
25                    G=255;
26                }
27                bmOut.setPixel(x, y, Color.argb(A, R, G, B));
28            }
29        }
30        return bmOut;
31    }
```

2.7.2 Rezultat



Sepia ze współczynnikiem W=20



Sepia ze współczynnikiem W=40

2.8 Krok 8: Detekcji krawędzi za pomocą krzyża Robertsa

Detekcja krawędzi za pomocą krzyża Robertsa polega na w pierwszym kroku na konwolucji z dwoma maskami 2x2

$$G_1 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

oraz

$$G_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Krok drugi polega na zamianie pola wektorowego na pole skalarne korzystając ze wzoru:

$$I(x,y) = \sqrt{G_1^2 + G_2^2} \tag{1}$$

2.8.1 Czynności

Metoda realizująca filtr Robertsa:

```
1     public Bitmap Roberts(Bitmap src) {
2         Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
              ;
3         int height = src.getHeight();
4         int width = src.getWidth();
5         for (int y = 3; y < height-3; y++) {
6             for (int x = 3; x < width-3; x++) {
7                 int p3=Color.red(src.getPixel(x + 1, y));
8                 int p4=Color.red(src.getPixel(x + 1, y + 1));
9                 int p5=Color.red(src.getPixel(x, y + 1));
10
11                int gg = Math.abs(Color.red(src.getPixel(x,y))- p4)+Math.abs(p3-p5);
12                if(gg > 255) gg = 255;
13                bmOut.setPixel(x, y, Color.rgb(gg, gg, gg));
14            }
15        }
16        return bmOut;
17    }
```

2.8.2 Rezultat

Obrazek po filtrze Robertsa

2.9 Krok 9: Filtr Sobela

Pierwszym krokiem algorytmy jest konwolucja z dwoma maskami 3x3:

$$G_1 = \begin{pmatrix} -1 & 0 & 0 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

oraz

$$G_2 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Następnie stosując poniższy wzór wykonujemy zamianę pola wektorowego na skalarne:

$$I(x,y) = \sqrt{G_1^2 + G_2^2} \tag{2}$$

2.9.1 Czynności

Metoda realizująca filtr Sobela

```
1   public Bitmap Sobel(Bitmap src) {
2         Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
              ;
3         int height = src.getHeight();
4         int width = src.getWidth();
5         for (int y = 3; y < height-3; y++) {
6             for (int x = 3; x < width-3; x++) {
7                       int p0=Color.red(src.getPixel(x - 1, y - 1));
8                       int p1=Color.red(src.getPixel(x, y - 1));
9                       int p2=Color.red(src.getPixel(x + 1, y - 1));
10                      int p3=Color.red(src.getPixel(x + 1, y));
11                      int p4=Color.red(src.getPixel(x + 1, y + 1));
```

```
                        int p5=Color.red(src.getPixel(x, y + 1));
                        int p6=Color.red(src.getPixel(x - 1, y + 1));
                        int p7=Color.red(src.getPixel(x - 1, y));
                        int xxg = ((p2+2*p3+p4)-(p0+2*p7+p6));
                        int yyg = ((p6+2*p5+p4)-(p0+2*p1+p2));
                        int g = (int) Math.hypot(xxg,yyg);
                        if(g > 255) g = 255;
                        bmOut.setPixel(x, y, Color.rgb(g, g, g));
                }
            }
        return bmOut;
    }
```

2.9.2 Rezultat

Obrazek po filtrze Sobela

2.10 Krok 10: Wyrównanie histogramu dla obrazu kolrowego i w odcieniach szarości

Operacja ta polega na poprawianiu kontrastu analizowanego obrazu z wykorzystaniem jego histogramu.Ponieważ w operacjach bezkontaksotwych: "Wszystkie piksele o jednakowej intensywności są traktowane identycznie" więc nie warto wyznaczać nowej wartości piksela za każdym razem. Dlatego dla zoptymalizowania algorytmu wykrozystałam tablicę LUT.

2.10.1 Czynności

Wyrównanie histogramu dla obrazu kolrowego

```
public Bitmap HistoKol(Bitmap src) {
        Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
            ;
        int height = src.getHeight();
        int width = src.getWidth();
```

```
5           int l_arrRed[] = new int[256];
6           int l_arrGreen[] = new int[256];
7           int l_arrBlue[] = new int[256];
8           for (int x = 0; x < width; x++) {
9               for (int y = 0; y < height; y++) {
10                  l_arrRed[Color.red(src.getPixel(x, y))]++; // zlicza piksele w jednym
                        kolorze
11                  l_arrGreen[Color.green(src.getPixel(x, y))]++;
12                  l_arrBlue[Color.blue(src.getPixel(x, y))]++;
13              }
14          }
15          double w=width*height;
16
17          double[] DRed= new double[256];
18          for( int i =0; i < 256; i++)
19          {
20              int zmienna=l_arrRed[0];
21              for( int k=0; k <i+1; k++)
22              {
23                  zmienna+=l_arrRed[k];
24              }
25              System.out.println(zmienna);
26
27              DRed[i] = zmienna/w;
28          }
29
30          double[] DGreen= new double[256];
31          for( int i =0; i < 256; i++)
32          {
33              int zmienna=l_arrRed[0];
34              for( int k=0; k <i+1; k++)
35              {
36                  zmienna+=l_arrRed[k];
37              }
38              DGreen[i] = zmienna/w;
39          }
40
41          double[] DBlue= new double [256];
42          for( int i =0; i < 256; i++)
43          {
44              int zmienna=l_arrRed[0];
45              for( int k=0; k <i+1; k++)
46              {
47                  zmienna+=l_arrRed[k];
48              }
49              DBlue[i] = zmienna/w;
50          }
51
52          int n = 0;
53          while(DRed[n]<=0)
54          {
55              n=n+1;
56          }
57          double minDRed = DRed[n];
58
59          int k = 0;
60          while(DBlue[k]<=0)
61          {
62              k=k+1;
63          }
64          double minDBlue = DBlue[k];
65
66          int j = 0;
67          while(DGreen[j]<=0)
68          {
69              j=j+1;
70          }
71          double minDGreen = DGreen[j];
```

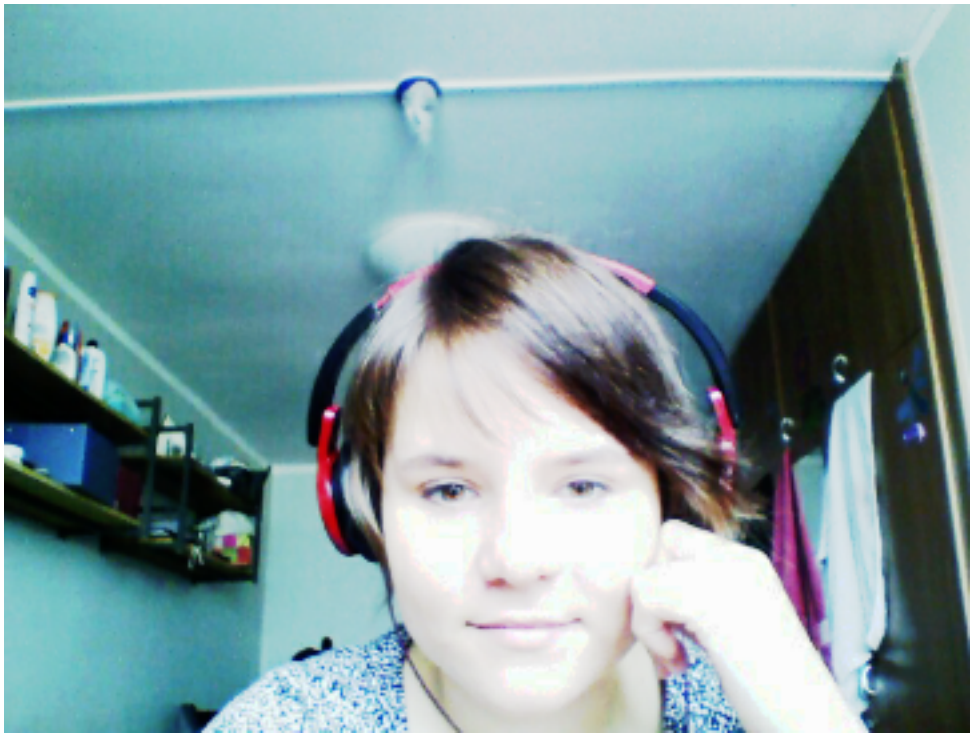
```
72
73
74          int[] lutR = new int[256];
75          int[] lutG = new int[256];
76          int[] lutB = new int[256];
77          for(int i = 0 ;i<256;++i){
78              lutR[i]=(int) Math.floor(((DRed[i]-minDRed)/(1-minDRed))*255);
79              lutG[i]=(int) Math.floor(((DGreen[i]-minDGreen)/(1-minDGreen))*255);
80              lutB[i]=(int) Math.floor(((DBlue[i]-minDBlue)/(1-minDBlue))*255);
81
82          }
83          int r, g, b;
84          for (int x = 0; x <width; x++) {
85              for (int y = 0; y < height; y++) {
86                  r = lutR[Color.red(src.getPixel(x, y))];
87                  g = lutG[Color.green(src.getPixel(x, y))];
88                  b = lutB[Color.blue(src.getPixel(x, y))];
89                  bmOut.setPixel(x, y, Color.rgb(r, g, b));
90              }
91          }
92          // return final bitmap
93          return bmOut;
94      }
```

Wyrównanie histogramu dla obrazu w odcieniach szarości.

```
1       public Bitmap HistoGray(Bitmap src) {
2           Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
                ;
3           int height = src.getHeight();
4           int width = src.getWidth();
5           int l_arr[] = new int[256];
6           for (int x = 0; x < width; x++) {
7               for (int y = 0; y < height; y++) {
8                   l_arr[Color.red(src.getPixel(x, y))]++; // zlicza piksele w jednym kolorze
9               }
10          }
11          double w=width*height;
12
13          double[] D= new double[256];
14          for( int i =0; i < 256; i++)
15          {
16              int zmienna=l_arr[0];
17              for( int k=0; k <i+1; k++)
18              {
19                  zmienna+=l_arr[k];
20              }
21              System.out.println(zmienna);
22
23              D[i] = zmienna/w;
24          }
25          int n = 0;
26          while(D[n]<=0)
27          {
28              n=n+1;
29          }
30          double minD = D[n];
31
32          int[] lut = new int[256];
33          for(int i = 0 ;i<256;++i){
34              lut[i]=(int) Math.floor(((D[i]-minD)/(1-minD))*255);
35          }
36          int z;
37          for (int x = 0; x <width; x++) {
38              for (int y = 0; y < height; y++) {
39                  z = lut[Color.red(src.getPixel(x,y))];
40                  bmOut.setPixel(x, y, Color.rgb(z, z, z));
```

```
41                  }
42              }
43              // return final bitmap
44              return bmOut;
45          }
```

2.10.2 Rezultat



Wyrównanie histogramu dla obrazka kolorowego

Wyrównanie histogramu dla obrazka w odcieniach szarości

2.11 Krok 11: Wykrywanie skóry

Wykrywanie skory mozna wykonac na wiele sposobow. Ja w mojej aplikacji zaimplementowałam dwa z nich. Pierwszy polega na zastosowaniu poniższego algorytmu:

- Dla każdego piksela oryginalnego obrazka (koloroweg) usuwa się kanały czerwony i zielony.
- Wykonuje się operację odjęcia kanału zielonego od czerwonego.
- Jeśli uzyskany wynik jest ujemny, do nowego obrazka, w miejscu odpowiadającym analizowanemu pikselowi, podstawia się wartość 0 do wszystkich kanałów kolorów.
- W przeciwnym razie w miejscu odpowiadającym analizowanemu pikselowi, podstawia się wartość różnicy do wszystkich kanałów kolorów.
- W ten sposób uzyskuje się obraz w odcieniach szarości, który następnie binaryzuje się z progiem równym 30

2.11.1 Czynności

Kod metody realizującej wykrywanie skory

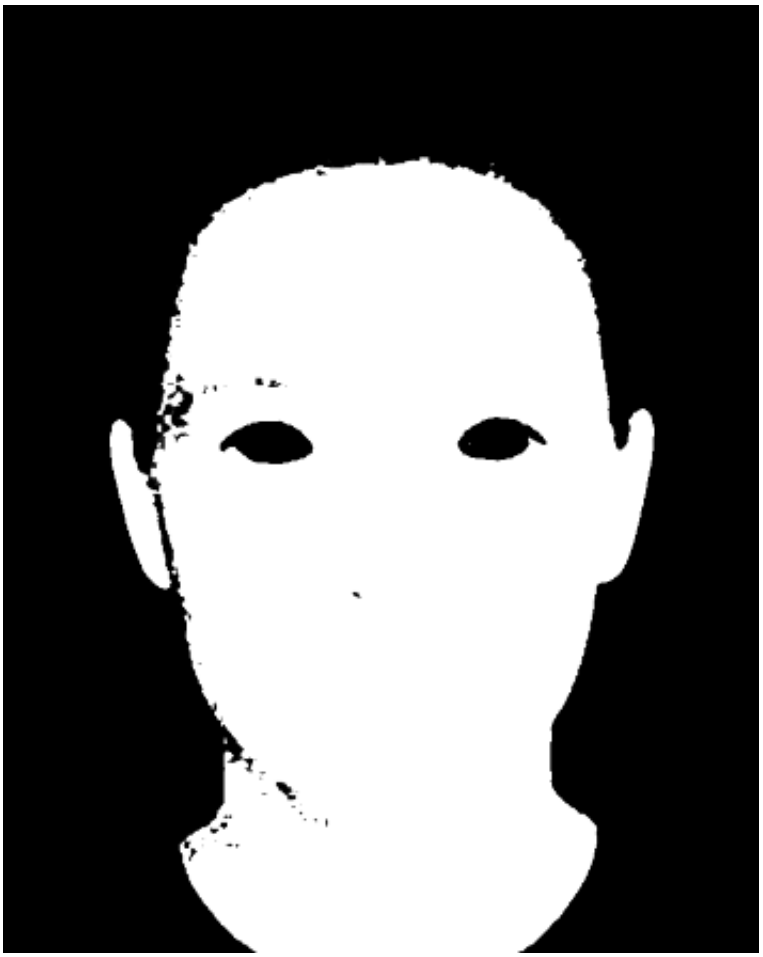
```
1   public Bitmap WykrywanieSkory1(Bitmap src) {
2       Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
            ;
3       int A, R, G, B;
4       int pixelColor;
5       int height = src.getHeight();
6       int width = src.getWidth();
7       for (int y = 0; y < height; y++) {
8           for (int x = 0; x < width; x++) {
9               pixelColor = src.getPixel(x, y);
10              A = Color.alpha(pixelColor);
11              R = Color.red(pixelColor)-Color.green(pixelColor);
12              G = Color.red(pixelColor)-Color.green(pixelColor);
13              B = Color.red(pixelColor)-Color.green(pixelColor);
14              if(R<0)
```

```
                {
                    R=B=G=0;
                }
                else if(R>255)
                {
                    R=B=G=255;
                }


                if(R>30 && G>30 && B>30)
                {
                    R=G=B=255;
                }
                if(R<30 && G<30 && B<30)
                {
                    R=G=B=0;
                }
                bmOut.setPixel(x, y, Color.argb(A, R, G, B));
            }
        }
        return bmOut;
    }
```

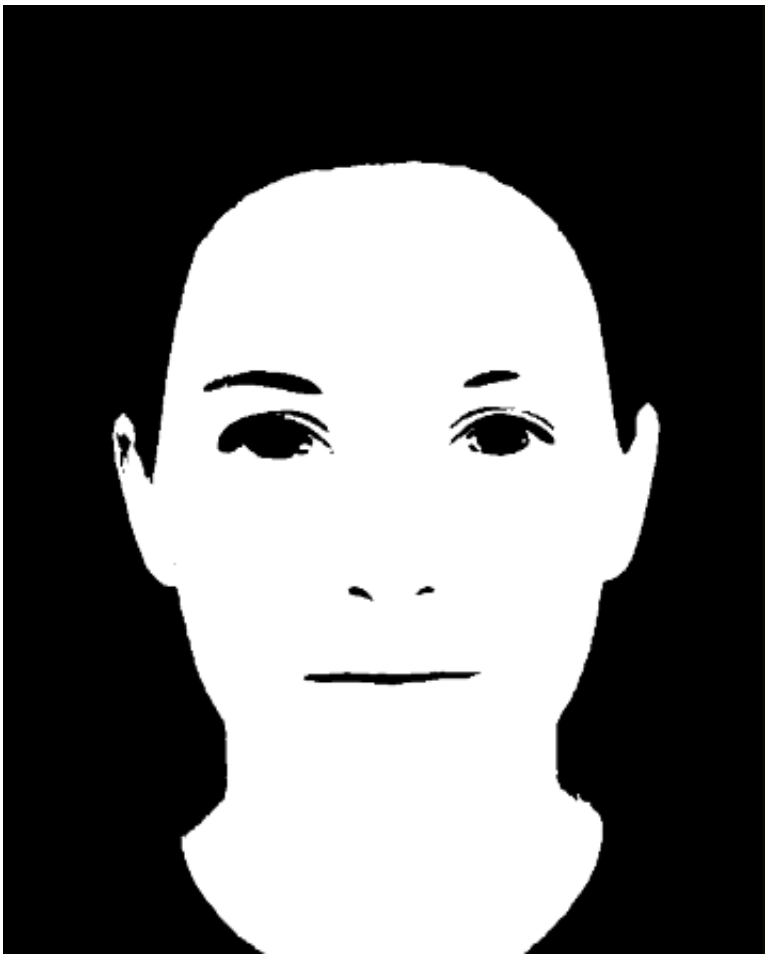
Algorytm drugi - polega na ustawieniu wartosci kanału R G i B na 255 dla pikseli spełniających określone warunki.

```

    public Bitmap WykrywanieSkory2(Bitmap src) {
        Bitmap bmOut = Bitmap.createBitmap(src.getWidth(), src.getHeight(), src.getConfig())
            ;
        int A, R, G, B;
        int pixelColor;
        int height = src.getHeight();
        int width = src.getWidth();

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                pixelColor = src.getPixel(x, y);
                A = Color.alpha(pixelColor);
                R = Color.red(pixelColor);
                G = Color.green(pixelColor);
                B = Color.blue(pixelColor);
                if(R>95&& G>40 && B>20 &&
                (Math.max(Math.max(R,G),B)-Math.min(Math.min(R,G),B))>15 &&
                (Math.abs(R-G))>15 && R>G && R>B)
                {
                    R=B=G=255;
                }
                else
                {
                    R=B=G=0;
                }
                bmOut.setPixel(x, y, Color.argb(A, R, G, B));
            }
        }
        return bmOut;
    }
```

2.11.2 Rezultat



Obrazek po zastosowaniu wykrywania skory algorytmem pierwszym



Obrazek po zastosowaniu wykrywania skory algorytmem drugim

3 Wnioski

Aplikacja wykonuje wszystkie operacje poprawnie.

Literatura

[1] `misztal.edu.pl`