LAPORAN AKHIR PROJECT UAS
PENGOLAHAN CITRA DIGITAL (TIF23)

Oleh:

Amanda Wulandari	(32180113)
Calvin Christopher	(32180011)
Edo Juliyanto	(32190008)
Hendrik	(32180001)
Yogi Wijaya	**(32180009)**

Teknik Informatika Fakultas Teknologi dan Desain

Universitas Bunda Mulia

2020

KATA PENGANTAR

Puji syukur kehadirat Tuhan yang Maha Esa yang telah memberikan Rahmat dan Anugrah-Nya sehingga kami dapat menyelesaikan project Ujian Akhir Semester (UAS) ini dengan baik dan tepat pada waktunya. Penulisan laporan ini untuk memenuhi tugas Bapak Teady Matius Surya Mulyana, S.Kom., M.Kom pada mata kuliah TIF23 – PENGOLAHAN CITRA DIGITAL. Laporan ini juga bertujuan untuk menambah wawasan tentang Pengolahan Citra Digital bagi para pembaca dan penulis.

Kami juga mengucapkan terima kasih kepada semua pihak yang telah membagikan sebagian pengetahuan, informasi, dan ilmunya sehingga kami dapat menyelesaikan tugas project Ujian Akhir Semester (UAS) ini. Kami menyadari Laporan yang kami tulis ini masih jauh dari kata sempurna. Oleh karena itu, kritik dan saran yang membangun akan kami nantikan demi kesempurnaan Laporan ini.

Jakarta, 25 November 2020

Penulis

DAFTAR ISI

BAB I
RUMUSAN MASALAH

1.1. Latar Belakang

Seiring dengan perkembangan zaman, pengetahuan dan teknologi berkembang sangat pesat. Salah satu teknologi yang mengalami perkembangan yang begitu pesat adalah Pengolahan Citra Digital.

Pengolahan Citra Digital adalah Teknik yang mempelajari pengolahan dan analisis semuah gambar pada komputerisasi. Pengolahan Citra Digital bertujuang untuk mengelola citra warna pada gambar untuk mengubah nilai-nilai pixel pada file bitmap gambar tesebut dengan banyak filter/cara untuk mengubah warna pada gambar tesebut sesuai dengan kebutuhan.

1.2. Rumusan Masalah

Pada project UAS ini terdapat beberapa rumusan masalah, antara lain sebagai berikut:

1. Bagaimana menciptakan sebuah ornament atau desain batik?
2. Bagaimana cara menggabungkan seluruh *code program* didalam satu button?

1.3. Pembagian Tugas

1. Amanda Wulandari : Filtering
2. Calvin Christoper : Deteksi Tepi
3. Edo Juliyanto : Grayscale dan Biner
4. Hendrik : Invers
5. Yogi Wijaya : Ornament

BAB II
LANDASAN TEORI

2.1. Pengolahan Citra Digital

Pengolahan Citra Digital (Digital Image Processing) merupakan bidang ilmu yang mempelajari tentang bagaimana suatu citra itu dibentuk, diolah, dan dianalisis sehingga menghasilkan informasi yang dapat dipahami oleh manusia.

2.2. Citra Grayscale

Jenis citra yang kedua adalah citra grayscale. Citra grayscale merupakan citra yang nilai intensitas pikselnya didasarkan pada derajat keabuan. Pada citra grayscale 8-bit, derajat warna hitam sampai dengan putih dibagi ke dalam 256 derajat keabuan di mana warna hitam sempurna direpresentasikan dengan nilai 0 dan putih sempurna dengan nilai 255. Citra RGB dapat dikonversi menjadi citra grayscale sehingga dihasilkan hanya satu kanal warna.

2.3. Citra Biner

Citra biner adalah citra yang pikselnya memiliki kedalaman bit sebesar 1 bit sehingga hanya memiliki dua nilai intensitas warna yaitu 0 (hitam) dan 1 (putih). Citra grayscale dapat dikonversi menjadi citra biner melalui proses thresholding. Dalam proses thresholding, dibutuhkan suatu nilai threshold sebagai nilai pembatas konversi. Nilai intensitas piksel yang lebih besar atau sama dengan nilai threshold akan dikonversi menjadi 1. Sedangkan nilai intensitas piksel yang kurang dari nilai threshold akan dikonversi menjadi 0. Misalnya nilai threshold yang digunakan adalah 128, maka piksel yang mempunyai intensitas kurang dari 128 akan diubah menjadi 0 (hitam) dan yang lebih dari atau sama dengan 128 akan diubah menjadi 1 (putih).

2.4. Smoothing

Smoothing adalah proses filter yang melewatkan komponen citra dengan nilai intensitas yang rendah dan meredam komponen citra dengan nilai intensitas yang tinggi. Smoothing akan menyebabkan citra menjadi lebih halus dan lebih blur.

2.5. Deteksi Tepi

Deteksi tepi berfungsi untuk mengidentifikasi garis batas (boundary) dari suatu objek yang terdapat pada citra. Tepian dapat dipandang sebagai lokasi piksel dimana terdapat nilai perbedaan intensitas citra secara ekstrem. Sebuah edge detector bekerja dengan cara mengidentifikasi dan menonjolkan lokasi-lokasi piksel yang memiliki karakteristik tersebut.

2.7 Invers

Invers adalah proses membalik nilai derajat keabuan dimana titik terang akan menjadi gelap dan titik gelap akan menjadi terang. Atau bisa juga disebut negatif. Untuk membalik nilai derajat keabuan digunakan rumus sebagai berikut: Sehingga jika sebuah titik memiliki nilai derajat keabuan 0 maka akan berubah menjadi 255. sebaliknya jika titik memiliki nilai 255 akan menjadi 0.xi=255−xg

2.8 Ornament

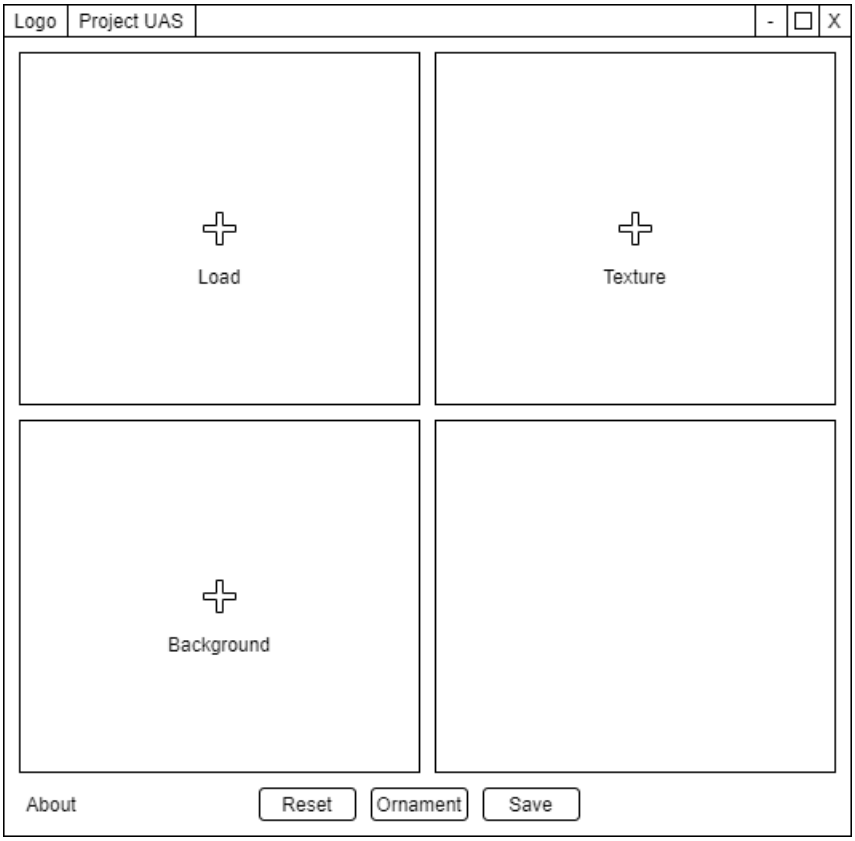
Ornament adalah gabungan dari 2 gambar yang telah dihitung nilai pixelnya yang membuat gambar terlihat hanya satu gambar. Gambar tersebut bermotif yang dapet dibuat menjadi motif baik dan sebagainya. Disini gambar yang diinput dari image 1 akan digabungkan dengan image 2 atau 3 yang menghasilkan perpaduan image 1 dan 2/3.

BAB III
RANCANGAN

3.1. Desain Mock Up

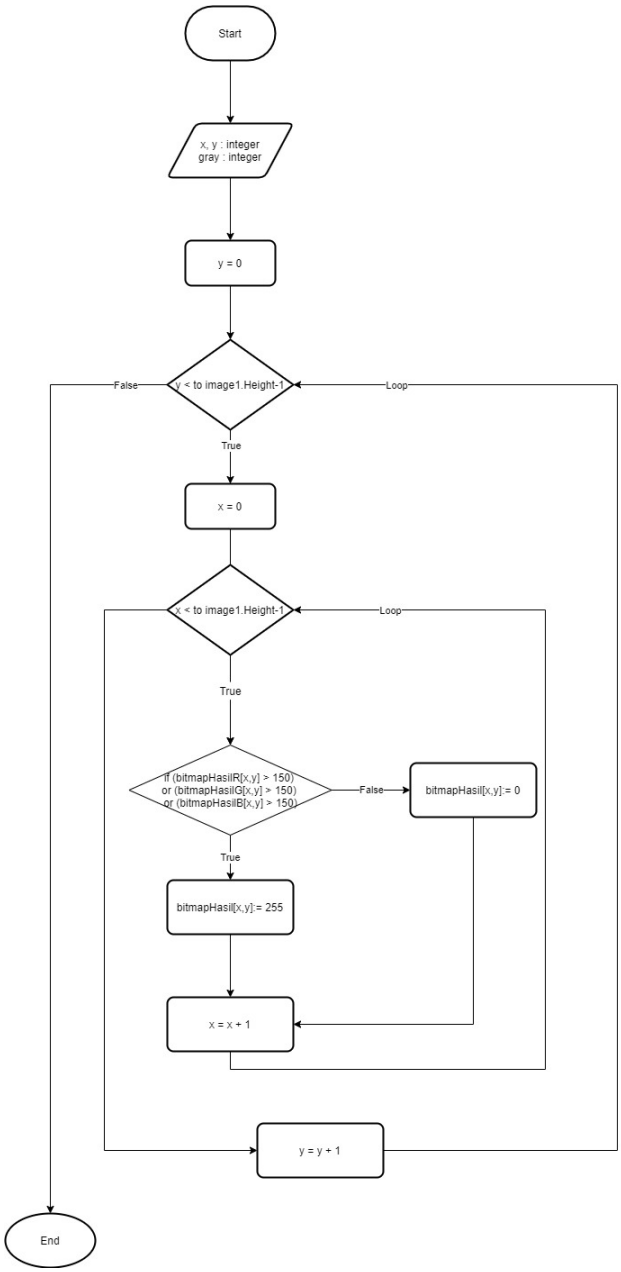
About:

Main:



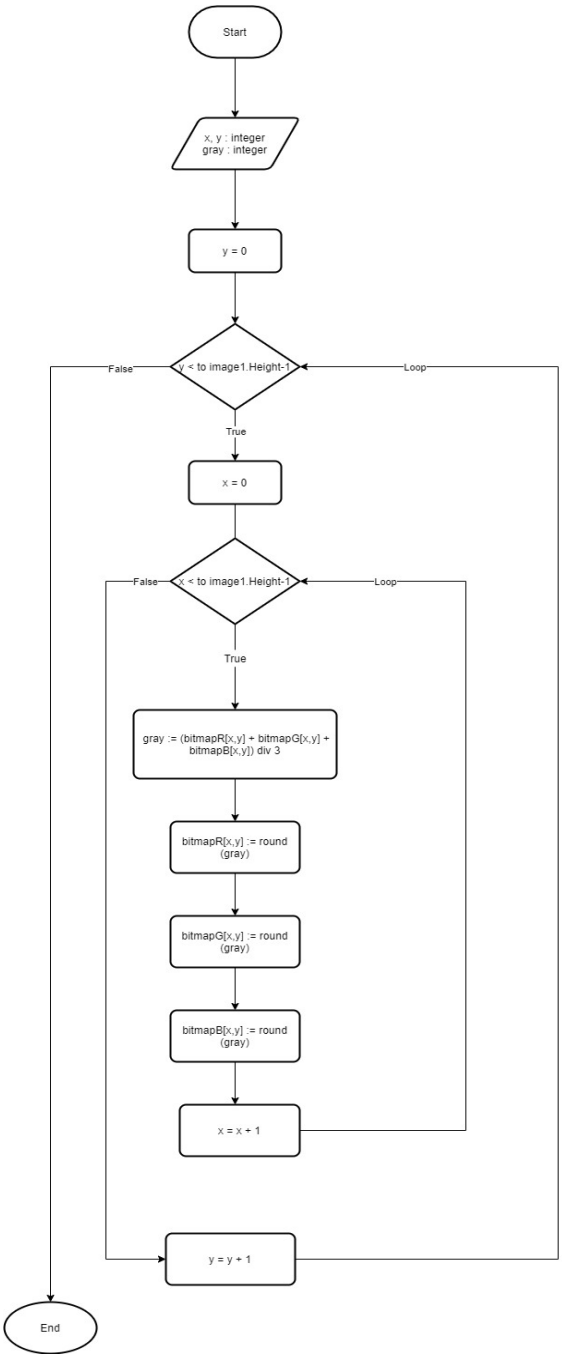
3.2. Desain Flowchart

3.2.1. Flowchart Binary



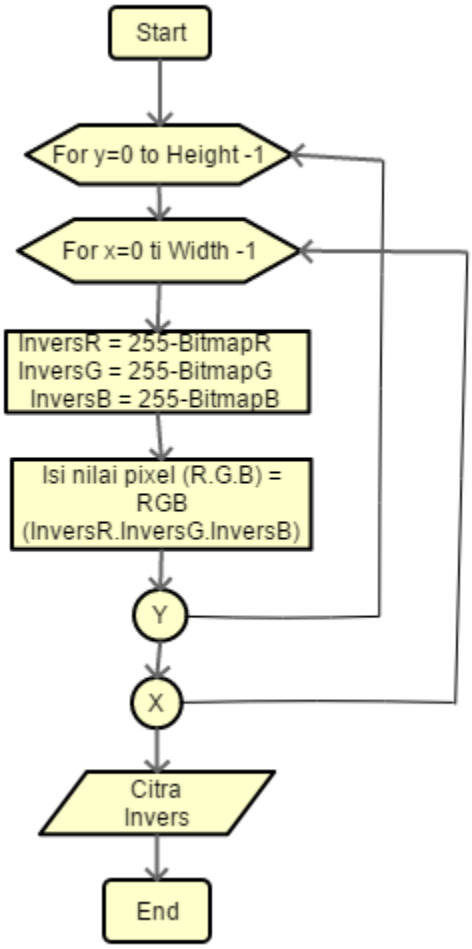
Berdasarkan flowchart diatas, pertama akan terjadi deklarasi variabel x dan y sebagai interger dan gray sebagai byte. Kemudian, menginisialisasikan nilai y = 0. Setelah itu, akan masuk looping y. selama nilai y < to image1.height-1, maka akan terjadi looping. Namun, jika nilai y tidak < to image1.height-1, maka tidak akan terjadi looping lagi dan akan terjadi end process. Berikutnya, misalkan masih terjadi looping, maka process akan berlanjut dimana akan terjadi inisialisasi lagi untuk x = 0. Setelah itu, akan masuk ke looping x. selama nilai x < to image1.height-1, maka akan terjadi looping. Namun, jika nilai x tidak < to image1.height-1, maka looping untuk x akan berhenti dan akan kembali ke looping y serta penambahan 1 nilai pada y. Berikutnya, jika masih terjadi looping pada looping x, maka akan masuk ke process pemilihan dimana jika bitmaphasilR, bitmaphasilG, bitmaphasilB >150, maka bitmapHasil akan bernilai 0, tetapi jika bitmaphasilR, bitmaphasilG, bitmaphasilB tidak >150, maka bitmaphasil akan bernilai 255 .Setelah selesai, akan masuk ke looping x lagi dan terjadi penambahan nilai pada x. process itu akan terjadi terus menerus hingga nilai x tidak < to image1.height-1.

3.2.2. Flowchart Grayscale



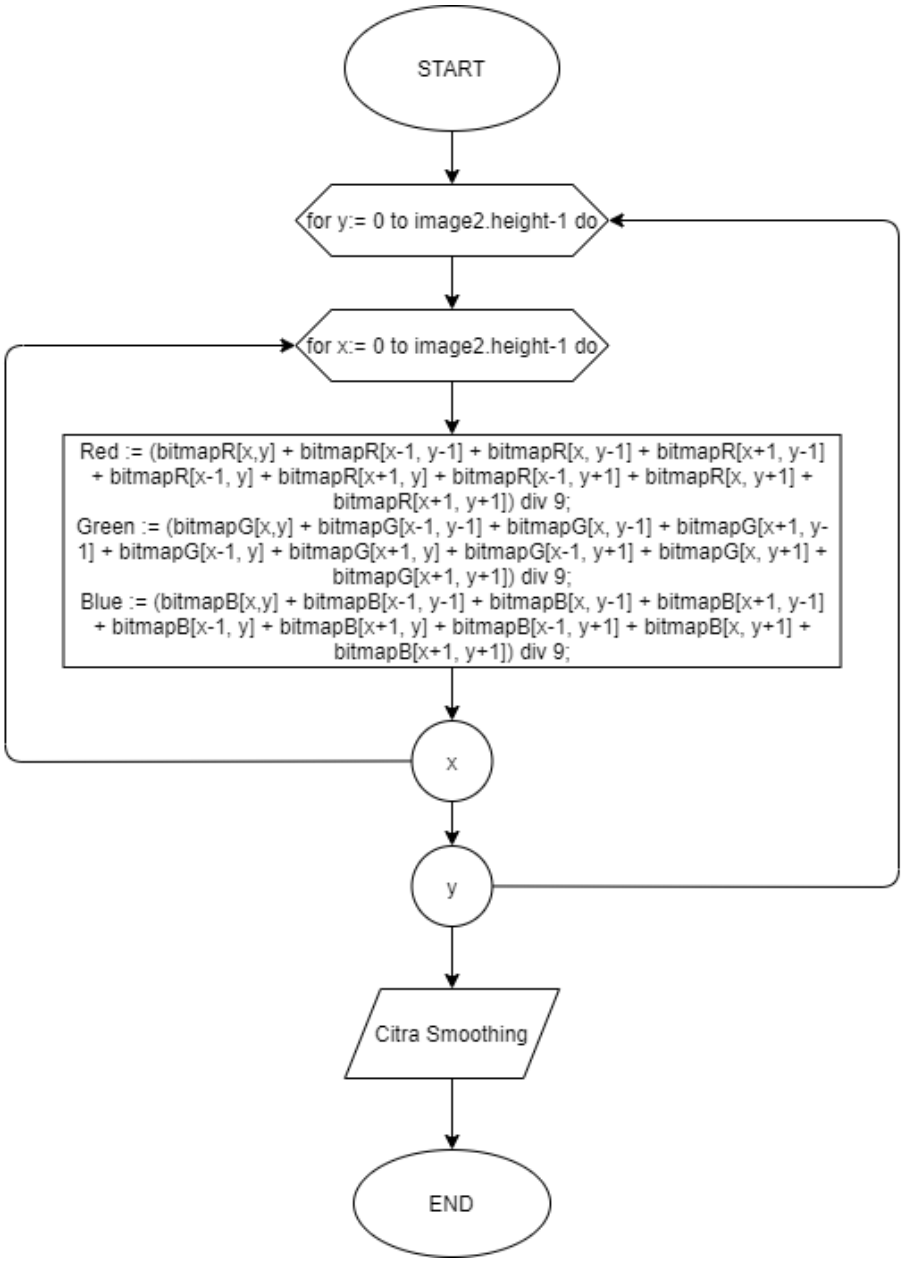
Berdasarkan flowchart diatas, pertama akan terjadi deklarasi variabel x dan y sebagai interger dan gray sebagai byte. Kemudian, menginisialisasikan nilai y = 0. Setelah itu, akan masuk looping y. selama nilai y < to image1.height-1, maka akan terjadi looping. Namun, jika nilai y tidak < to image1.height-1, maka tidak akan terjadi looping lagi dan akan terjadi end process. Berikutnya, misalkan masih terjadi looping, maka process akan berlanjut dimana akan terjadi inisialisasi lagi untuk x = 0. Setelah itu, akan masuk ke looping x. selama nilai x < to image1.height-1, maka akan terjadi looping. Namun, jika nilai x tidak < to image1.height-1, maka looping untuk x akan berhenti dan akan kembali ke looping y serta penambahan 1 nilai pada y. Berikutnya, jika masih terjadi looping pada looping x, maka akan masuk ke process mencari nilai grayscale. Setelah itu, akan masuk proses dimana setiap nilai dari bitmapR, bitmapG dan bitmapB akan digantikan dengan nilai grayscale yang telah dicari. Setelah selesai, akan masuk ke looping x lagi dan terjadi penambahan nilai pada x. process itu akan terjadi terus menerus hingga nilai x tidak < to image1.height-1.

3.2.3. Flowchart Invers



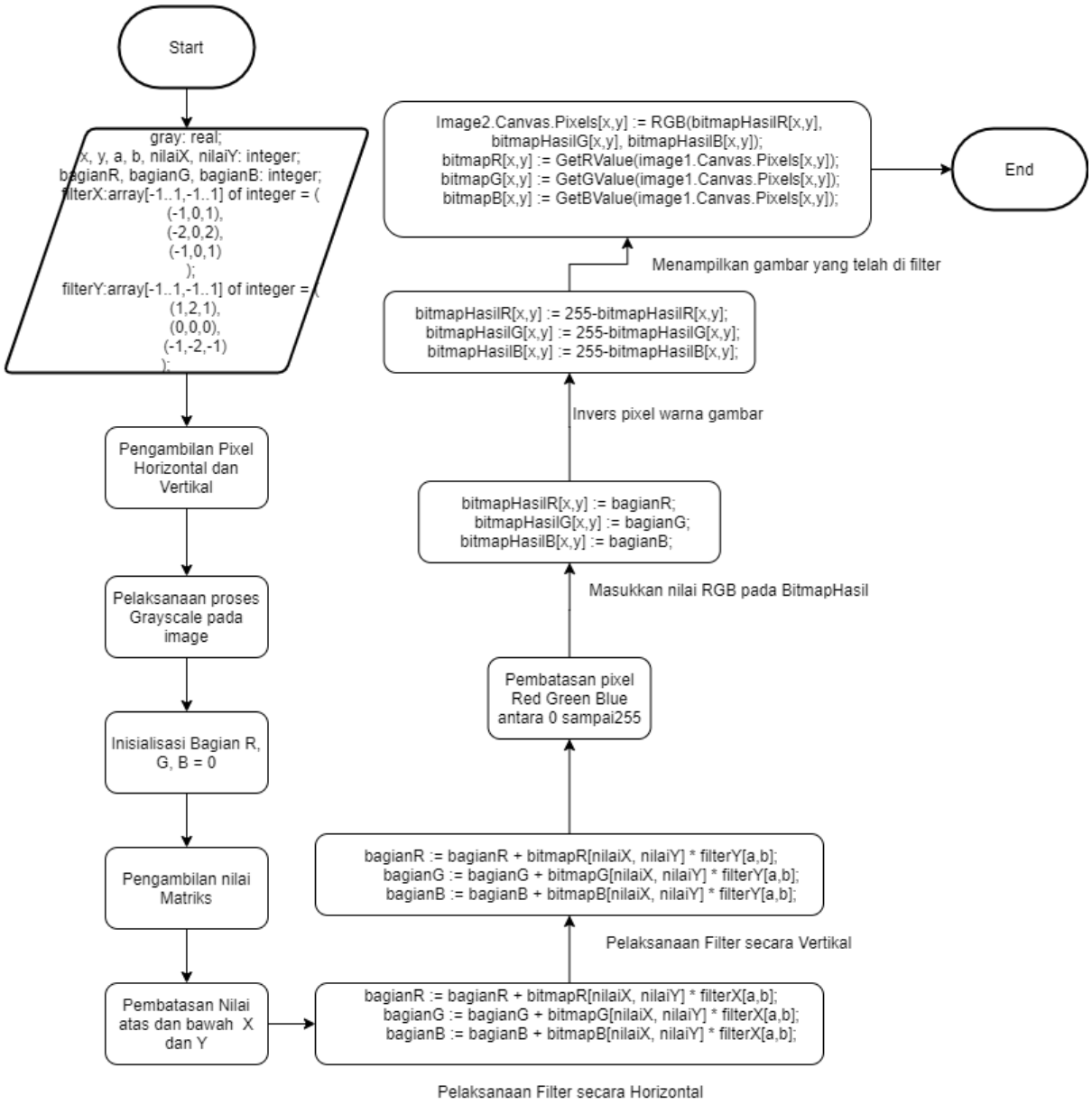
Berdasarkan flowchart diatas, pertama akan terjadi deklarasi variabel x dan y sebagai interger dan Invers R,G,B sebagai byte. Kemudian, menginisialisasikan nilai y = 0. Setelah itu, akan masuk looping y. selama nilai y < to image1.height-1, maka akan terjadi looping. Namun, jika nilai y tidak < to image1.height-1, maka tidak akan terjadi looping lagi dan akan terjadi end process. Berikutnya, misalkan masih terjadi looping, maka process akan berlanjut dimana akan terjadi inisialisasi lagi untuk x = 0. Setelah itu, akan masuk ke looping x. selama nilai x < to image1.height-1, maka akan terjadi looping. Namun, jika nilai x tidak < to image1.height-1, maka looping untuk x akan berhenti dan akan kembali ke looping y serta penambahan 1 nilai pada y. Berikutnya, jika masih terjadi looping pada looping x, maka akan masuk ke process invers image. Setelah itu, akan masuk proses dimana setiap nilai invers R,G,B = 255-Bitmap R,G,B dan akan menghasilkan pixel gelap akan menjadi putih dan pixel putih akan menjadi gelap sampai selesai.

3.2.4. Flowchart Smoothing



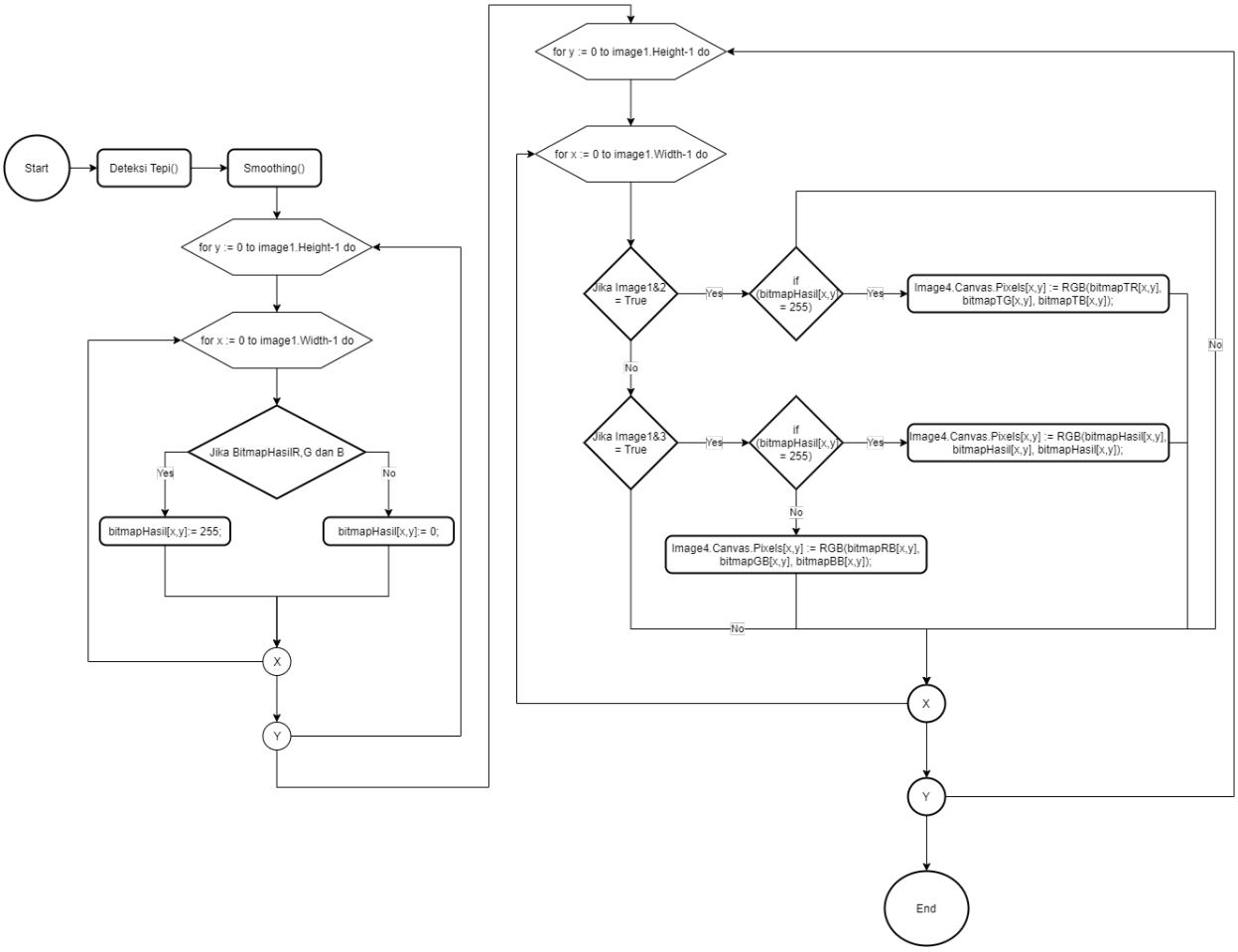
Smoothing yang berarti memperhalus mempunyai tujuan untuk menekan/mengurangi gangguan (noise) yang ada pada citra. Smoothing ini menggunakan metode mean filter dan terdapat perhitungan matriks di dalamnya. Metode mean filtering akan menjumlahkan nilai piksel dengan ketetanggaannya lalu dibagi untuk mendapatkan nilai rata-ratanya. Hasil dari smoothing sebetulnya kebalikan dari sharpening. Dimana sharpening akan membuat piksel-piksel terlihat lebih kasar sedangkan smoothing terlihat lebih halus.

3.2.5. Flowchart Deteksi Tepi



Process dari deteksi tepi, berarti dari image yang sudah bertipe piksel, akan dilakukan process grayscale kemudia diambil garis garis tegas dari sebuah gambar sehingga hasilnya berupa tepi tepi dari obek gambar. Metode yang digunakan dalam metode deteksi tepi adalah **Metode Sobell.** Process filter yang digunakan ada 2 jenis yaitu secara horizontal dan vertikal agar hasil yang didapat lebih maksimal. Process filter dilakukan pada bagian warna gambar Red Green dan Blue, kemudian dilakukan pembatasan warna yang sudah difilter agar hasil warna tidak melebihi aturan warma 8 bit.

3.2.6. Flowchart Ornament



Di sini saya memanggil deteksi tepi dan smoothing lalu dimasukkan ke binerisasi untung menampung hasilnya 255/0 tidak ada angka lain. Lalu akan di cek jika image1&2 yang load/true akan menjalankan fungsi jika bitmapHasil 255 makan image4 akan diisi oleh nilai bitmapTR,TG dan TB. Sedangakan jika iamge 1&3 yang di load/true maka berjalan fungsi jika bitmaphasil 255 image4 akan diisi nilai ButmapHasil yang telah dihitung di deteksi tepi. Sedangkan jika bitmap hasil adalah 0. Maka image 4 akan diisi bitmapRB,RG dan BB.

BAB IV
IMPLEMENTASI

4.1. Code Program

4.1.1. Code Binary

```
var
  x, y : integer;
  gray : byte;
begin
  for y := 0 to image1.Height-1 do
  begin
    for x := 0 to image1.Width-1 do
    begin
      if (bitmapHasilR[x,y] > 150) or (bitmapHasilG[x,y] > 150) or (bitmapHasilB[x,y] > 150) then
      begin
       bitmapHasil[x,y]:= 255;
      end
      else
      begin
       bitmapHasil[x,y]:= 0;
      end
    end;
  end;
end;
```

Penjelasan Code:

Citra biner adalah citra yang pikselnya memiliki kedalaman bit sebesar 1 bit sehingga hanya memiliki dua nilai intensitas warna yaitu 0 (hitam) dan 1 (putih). Citra grayscale dapat dikonversi menjadi citra biner melalui proses thresholding. Dalam proses thresholding, dibutuhkan suatu nilai threshold sebagai nilai pembatas konversi. Nilai intensitas piksel yang lebih besar nilai threshold akan dikonversi menjadi 1. Sedangkan nilai intensitas piksel yang kurang dari atau sama dengan nilai threshold akan dikonversi menjadi 0. Misalnya nilai threshold yang digunakan adalah 128, maka piksel yang mempunyai intensitas kurang dari atau sama dengan 128 akan diubah menjadi 0 (hitam) dan yang lebih dari 128 akan diubah menjadi 1 (putih).

Pada program saya diatas, hampir sama seperti proses grayscale pertama-tama saya mendeklarkan variabel yang akan saya gunakan dalam proses grayscale ini. Dalam pendeklaran, saya mendeklar variabel x dan y dengan tipe data integer dan variabel gray dengan tipe data byte. Variabel x dan y tersebut sebagai nilai piksel dalam koordinat x dan koordinat y. Variabel gray sebgai variabel penampung nilai-nilai piksel yang sudah dihitung yang akan menjadi nilai grayscale dari citra tersebut. Kemudian, saya menggunakan perulangan for yang pertama untuk menentukan nilai piksel citra di koordinat y dan for yang kedua untuk menentukan nilai piksel citra di koordinat x. Kedua for itu digunakan untuk untuk mengubah setiap pixel yang ada didalam citra tersebut, perubahan ini dilakukan satu per satu agar seluruh citra dapat diubah seluruh pixelnya. Setelah itu, nilai-nilai yang didapatkan akan diproses dan akan disimpan hasil perhitungannya dalam varibel gray. Dalam proses tersebut setiap nilai piksel, baik nilai piksel pada citra Red, Green atau Blue. Caranya, tiap nilai piksel RGB tersebut ditambah dan di div dengan 3. Untuk lebih jelasnya, berikut rumusnya :

gray = (bitmapR[x,y] + bitmapG[x,y] + bitmapB[x,y]) div 3

Misalnya, suatu gambar memiliki nilai piksel R= 50, G = 20, B = 90. Maka, nilai gray nya adalah gray = 50 + 20 + 90 div 3, didapatkan nilai grayscale = 160 div 3 yaitu 53.333333333. Nantinya, nilai dari grayscale tersebut digunakan sebagai nilai pikses Red, Green, dan Blue. Sampai sini, kita sudah mendapatkan citra grayscalenya. Setelah itu, agar

dapat menjadi citra biner, kita lakukan proses tresholding pada citra grayscale tersebut. Untuk proses tersholding pada citra grayscalenya, sesuai dengan program diatas kita beri pembatas nilai, yaitu jika nilai gray nya kurang atau sama dengan dari nilai treshold yang ditentukan maka akan diubah ke citra biner hitam (0), sedangkan jika lebih dari nilai treshold yang ditentukan maka akan diubah ke citra biner putih (1). Misalnya nilai threshold yang digunakan adalah 128, maka piksel yang mempunyai intensitas kurang dari atau sama dengan 128 akan diubah menjadi hitam (0) dan yang lebih dari 128 akan diubah menjadi biner putih (1). Disini letak perbedaan antara proses grayscale dan proses biner, jika pada proses grayscale hanya mencari nilai grayscale pada tiap-tiap nilai piksel citra, maka beda dengan proses biner yan setelah menemukan nilai grayscale akan diuabah atau konversi ke citra biner melalui proses tresholding. Terakhir, nilai-nilai piksel RGB tersebut akan ditampilkan dalam image1.canvas sesuai dengan nilai grayscale apakah lebih dari atau sama dengan atau kurang dari nilai treshold yang ditentukan. maka, setelah itu akan tertampil citra biner. Citra yang tertampil, akan disimpan citranya oleh user nantinya.

4.1.2. Code Grayscale

```
var
  x, y : integer;
  gray : byte;
begin
  for y:=0 to image1.Height-1 do
  begin
    for x:=0 to image1.Width-1 do
    begin
      gray := (bitmapR[x,y] + bitmapG[x,y] + bitmapB[x,y]) div 3;
      bitmapR[x,y]:= round(gray);
      bitmapG[x,y]:= round(gray);
      bitmapB[x,y]:= round(gray);
    end;
  end
end;
```

Penjelasan Code:

Citra grayscale merupakan citra yang nilai intensitas pikselnya didasarkan pada derajat keabuan. Pada citra grayscale 8-bit, derajat warna hitam sampai dengan putih dibagi ke dalam 256 derajat keabuan di mana warna putih sempurna direpresentasikan dengan nilai 255 dan hitam sempurna dengan nilai 0. Citra RGB dapat dikonversi menjadi citra grayscale.

Pada program saya diatas, pertama-tama saya mendeklarkan variabel yang akan saya gunakan dalam proses grayscale ini. Dalam pendeklaran, saya mendeklar variabel x dan y dengan tipe data integer dan variabel gray dengan tipe data byte. Variabel x dan y tersebut sebagai nilai piksel dalam koordinat x dan koordinat y. Variabel gray sebgai variabel penampung nilai-nilai piksel yang sudah dihitung yang akan menjadi nilai grayscale dari citra tersebut. Kemudian, saya menggunakan perulangan for yang pertama untuk menentukan nilai piksel citra di koordinat y dan for yang kedua untuk menentukan nilai piksel citra di koordinat x. Kedua for itu digunakan untuk untuk mengubah setiap pixel yang ada didalam citra tersebut, perubahan ini dilakukan satu per satu agar seluruh citra dapat diubah seluruh pixelnya. Setelah itu, nilai-nilai yang didapatkan akan diproses dan akan disimpan hasil perhitungannya dalam varibel gray. Dalam proses tersebut setiap nilai piksel, baik nilai piksel pada citra Red, Green atau Blue. Caranya, tiap nilai piksel RGB tersebut ditambah dan di div dengan 3. Untuk lebih jelasnya, berikut rumusnya :

gray = (bitmapR[x,y] + bitmapG[x,y] + bitmapB[x,y]) div 3

Misalnya, suatu gambar memiliki nilai piksel R= 50, G = 20, B = 90. Maka, nilai gray nya adalah gray = 50 + 20 + 90 div 3, didapatkan nilai grayscale = 160 div 3 yaitu 53.333333333. Nantinya, nilai dari grayscale tersebut digunakan sebagai nilai pikses Red, Green, dan Blue. Terakhir, nilai-nilai piksel RGB tersebut akan ditampilkan dalam image1.canvas. maka, disitu akan tertampil citra grayscale dan akan disimpan oleh user nantinya

4.1.3. Code Invers

```
procedure TFormUtama.ButtonInversClick(Sender: TObject);
var
  x,y : integer;
  inversR, inversG, inversB, invers : byte;
begin
  for y:=0 to image1.Height-1 do
  begin
    for x:=0 to image1.Width-1 do
    begin
      inversR := 255 - bitmapR[x,y];
      inversG := 255 - bitmapG[x,y];
      inversB := 255 - bitmapB[x,y];
      image2.Canvas.Pixels[x,y] := RGB(inversR, inversG, inversB);
      end
      else
        image2.Canvas.Pixels[x,y] := RGB(invers,invers,invers);
    end;
  end;
end:
```

1

2

3

4

Penjelasan Code:

1. Variabel X,Y untuk perulangan dan dekralasi inversR, inversG, inversB, invers sebagai byte.

2. Untuk menentukan lebar dan tinggi image

3.Pada bagian ini invers R, invers G, inversB nilainnya 255 dikurang bitmapR,bitmapG,bitmapB.

4. Dibagian adalah untuk menampilkan hasil dari invers.

❖ Code Invers

Invers adalah proses membuat citra gambar menjadi warna negatif. Proses dalam pembuatan citra invers adalah mengubah pixel (red,green,blue) menjadi pixel (inversR,inverG,inversB). Rumus dari pembuatan invers adalah inversR = 255 - R, inversG = 255 - G, inversB = 255 - B

4.1.4. Code Smoothing

```
procedure TForm1.smoothing();
var
 x, y, ts : integer;
begin
  ts := 1;
  for y:=0 to Image3.Height-1 do
  begin
   for x:=0 to Image3.Width-1 do
   begin
    bitmapRB[x,y] := (bitmapRB[x,y] + bitmapRB[x-ts, y-ts] + bitmapRB[x, y-ts] + bitmapRB[x+ts, y-ts] + bitmapRB[x-ts, y] + bitmapRB[x+ts, y] + bitmapRB[x-ts, y+ts] + bitmapRB[x, y+ts] + bitmapRB[x+ts, y+ts]) div 9;
    bitmapGB[x,y] := (bitmapGB[x,y] + bitmapGB[x-ts, y-ts] + bitmapGB[x, y-ts] + bitmapGB[x+ts, y-ts] + bitmapGB[x-ts, y] + bitmapGB[x+ts, y] + bitmapGB[x-ts, y+ts] + bitmapGB[x, y+ts] + bitmapGB[x+ts, y+ts]) div 9;
    bitmapBB[x,y] := (bitmapBB[x,y] + bitmapBB[x-ts, y-ts] + bitmapBB[x, y-ts] + bitmapBB[x+ts, y-ts] + bitmapBB[x-ts, y] + bitmapBB[x+ts, y] + bitmapBB[x-ts, y+ts] + bitmapBB[x, y+ts] + bitmapBB[x+ts, y+ts]) div 9;
   end;
  end;
end;
```

Penjelasan Code:

Smoothing ini menggunakan metode mean filter dan terdapat perhitungan matriks di dalamnya. Metode mean filtering akan menjumlahkan nilai piksel dengan ketetanggaannya lalu dibagi untuk mendapatkan nilai rata-ratanya.

Rumus Mean Filtering:

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Keterangan :

X = Nilai rata-rata (Mean)

i = Nilai Awal

n = Jumlah data i

x = Nilai ke -i

Mean filtering yang digunakan untuk efek smoothing ini merupakan jenis spatial filtering, yang dalam prosesnya mengikutsertakan piksel-piksel disekitarnya. Piksel yang akan diproses dimasukkan dalam sebuah matrik yang berdimensi N X N. Ukuran N ini tergantung pada kebutuhan, tetapi nilai N harus ganjil sehingga piksel yang diproses dapat diletakkan tepat ditengah matriks.

Piksel-piksel disekitar piksel *ts* yang akan diproses, proses pengambilan piksel dimulai dengan mengambil piksel yang akan diproses, disimpan dalam nilai *ts*. Kemudian diambil piksel-piksel sekitarnya sehingga matriks terisi penuh.

Selanjutnya kita jumlahkan semua nilai yang terdapat pada matrik tersebut. Hasil penjumlahan tersebut dibagi dengan jumlah titik yang terdapat pada matrik tersebut. Bilangan pembagi ini dapat diperoleh dari perkalian antara N X N. Maka hasil pembaginya adalah 9. Sembilan diperoleh dari hasil kali matrik 3 X 3. Hasil pembagian tersebut akan menggantikan nilai *ts*. Nilai *ts* yang baru akan ditampilkan pada layar monitor untuk menggantikan nilai *ts* yang lama.

Proses diatas adalah untuk menggambar grayscale (hitam-putih), untuk menggambar berwarna maka masing-masing titik terlebih dahulu ditentukan nilai warna merah (R), hijau (G), dan biru (B). Masing-masing nilai RGB dijumlahkan. Hasil penjumlahan nilai RGB dibagi dengan jumlah titik yang diproses. Hasil dari pembagian digunakan untuk menentukan warna baru yang akan di letakkan pada titik *ts*.

4.1.5. Code Deteksi Tepi

```
procedure TForm1.deteksitepi();
var
 gray: real;
 x, y, a, b, nilaiX, nilaiY: integer;
 bagianR, bagianG, bagianB: integer;
 filterX:array[-1..1,-1..1] of integer = (
  (-1,0,1),
  (-2,0,2),
  (-1,0,1)
 );
 filterY:array[-1..1,-1..1] of integer = (
  (1,2,1),
  (0,0,0),
  (-1,-2,-1)
```

```
  );
begin
  for y:=0 to image1.Height-1 do
  begin
    for x:=0 to image1.Width-1 do
    begin
      gray := (bitmapR[x,y] + bitmapG[x,y] +
bitmapB[x,y]) div 3;
      bitmapR[x,y]:= round(gray);
      bitmapG[x,y]:= round(gray);
      bitmapB[x,y]:= round(gray);
    end;
  end;


  for y:=0 to image1.Height-1 do
  begin
    for x:=0 to image1.Width-1 do
    begin
      bagianR := 0;
      bagianG := 0;
      bagianB := 0;

      for a:=-1 to 1 do
      begin
        for b:=-1 to 1 do
        begin
          nilaiX := a + x ;
          nilaiY := b + y ;

          if nilaiX > image1.Width-1 then
          begin
            nilaiX := image1.Width-1;
          end;

          if nilaiX < 0 then
          begin
            nilaiX := 0;
          end;

          if nilaiY > image1.Height-1 then
          begin
            nilaiY := image1.Height-1;
          end;

          if nilaiY < 0 then
          begin
            nilaiY := 0;
          end;

          bagianR := bagianR + bitmapR[nilaiX, nilaiY] *
filterX[a,b];
          bagianG := bagianG + bitmapG[nilaiX, nilaiY] *
filterX[a,b];
          bagianB := bagianB + bitmapB[nilaiX, nilaiY] *
filterX[a,b];

          bagianR := bagianR + bitmapR[nilaiX, nilaiY] *
filterY[a,b];
          bagianG := bagianG + bitmapG[nilaiX, nilaiY] *
filterY[a,b];
          bagianB := bagianB + bitmapB[nilaiX, nilaiY] *
filterY[a,b];
        end;
      end;

      if bagianR > 255 then
      begin
        bagianR := 255;
      end;

      if bagianR < 0 then
      begin
        bagianR := 0;
      end;

      if bagianG > 255 then
      begin
        bagianG := 255;
```

```
    end;

    if bagianG < 0 then
    begin
      bagianG := 0;
    end;

    if bagianB > 255 then
    begin
      bagianB := 255;
    end;

    if bagianB < 0 then
    begin
      bagianB := 0;
    end;

    bitmapHasilR[x,y] := bagianR;
    bitmapHasilG[x,y] := bagianG;
    bitmapHasilB[x,y] := bagianB;
      end;
    end;
    for y:=0 to image1.Height-1 do
    begin
      for x:=0 to image1.Width-1 do
      begin
        bitmapHasilR[x,y] := 255-bitmapHasilR[x,y];
        bitmapHasilG[x,y] := 255-bitmapHasilG[x,y];
        bitmapHasilB[x,y] := 255-bitmapHasilB[x,y];
      end;
    end;
    for y:=0 to image1.Height-1 do
    begin
      for x:=0 to image1.Width-1 do
      begin
        Image2.Canvas.Pixels[x,y] :=
    RGB(bitmapHasilR[x,y], bitmapHasilG[x,y],
    bitmapHasilB[x,y]);
    end;
    end;
    end;
```

Penjelasan Code:

Dalam proses Deteksi Tepi, saya menggunakan beberapa variabel yaitu

x, y sebagai integer untuk menampung lebar dan tinggi TImage

bagianR, bagianG, bagianB sebagai integer untuk mengambil nilai r, g, b setelah Filter

a, b sebagai integer untuk menampung integer matriks Filter

nilaiX, nilai Y sebagai integer untuk mengambil nilai x dan y yang akan di Filter

FilterX, Filter sebagai array of Integer untuk proses filter **(Metode Sobell)**

Langkah pertama yang saya lakukan adalah untuk mengambil nilai pixel per pixel pada gambar dengan menggunakan rumus

for y:=0 to Image2.height-1 do untuk pixel vertical

for x:=0 to Image2.width-1 do untuk pixel horizontal

Kemudian melakuka proses grayscale untuk mengubah gambar yang berwarna menjadi monochrome untuk memudahkan dalam pelaksanaan deteksi tepi dengan rumus

gray := (bitmapR[x,y] + bitmapG[x,y] + bitmapB[x,y]) div 3;

bitmapR[x,y]:= round(gray);

bitmapG[x,y]:= round(gray);

bitmapB[x,y]:= round(gray);

Setelah itu melakukan inisialisasi nilai R, G, B pada gambar dengan rumus

bagianR := 0;

bagianG := 0;

bagianB := 0;

Kemudian melakukan pemasukkan nilai x dan y pada gambar dengan matriks filter dengan rumus :

```
for a:=-1 to 1 do
    begin
     for b:=-1 to 1 do
     begin
       nilaiX := a + x ;
       nilaiY := b + y ;
```

Lalu, diberikan Batasan atas dan bawah pada masing – masing nilai X dan Nilai Y agar nilai X dan Y yang dihasilkan tidak melebihi dari pixel X dan pixel Y dari TImage1.

Setelah itu, Melakukan filter secara horizontal dan vertical untuk didapatkan tepi yang diinginkan . Metode yang digunakan dalam filter adalah **metode sobell** dikarenakan setelah

dilakukan percobaan dengan metode metode lain hasil yang didapatkan tidak sebagus dari metode Sobell. Proses filter dilakukan dengan rumus :

Filter Horizontal :

```
bagianR := bagianR + bitmapR[nilaiX, nilaiY] * filterX[a,b];
    bagianG := bagianG + bitmapG[nilaiX, nilaiY] * filterX[a,b];
    bagianB := bagianB + bitmapB[nilaiX, nilaiY] * filterX[a,b];
```

Filter Vertikal :

```
    bagianR := bagianR + bitmapR[nilaiX, nilaiY] * filterY[a,b];
    bagianG := bagianG + bitmapG[nilaiX, nilaiY] * filterY[a,b];
    bagianB := bagianB + bitmapB[nilaiX, nilaiY] * filterY[a,b];
```

Melakukan pemberian Batasan Atas dan bawah terhadap masinng masing BagianR, BagianG, BagianB agar warna yang dihasilkan tidak melebihi dari aturan 8 bit yang digunakan dalam gambar bertipe bitmap.

Kemudian memasukkan masing masing hasil filter kepada bitmapHasil dengan rumus :

```
bitmapHasilR[x,y] := bagianR;
  bitmapHasilG[x,y] := bagianG;
  bitmapHasilB[x,y] := bagianB;
```

Setelah itu, melakukan proses invers untuk membalikkan warna pada background dan pada objek agar hasil yang didapatkan bisa lebih akurat dan sesuai dengan gambar aslinya. Proses dilakukan dengan rumus :

```
bitmapHasilR[x,y] := 255-bitmapHasilR[x,y];
  bitmapHasilG[x,y] := 255-bitmapHasilG[x,y];
  bitmapHasilB[x,y] := 255-bitmapHasilB[x,y];
```

Dan yang terakhir dilakukan penampilan gambar pada TImage2 dengan memasukkan pixel pixel warna yang telah dilakukan filter kepada TImage2 sehingga dihasilkan gambar yang telah terdetksi tepinya. Penampilan gambar dilakukan dengan rumus :

for y:=0 to image1.Height-1 do

begin

for x:=0 to image1.Width-1 do

begin

Image2.Canvas.Pixels[x,y] := RGB(bitmapHasilR[x,y], bitmapHasilG[x,y], bitmapHasilB[x,y]);

end;

end;

4.1.6. Code Ornament

```
procedure
TForm1.ButtonOrnamentClick(Sender:
TObject);
var
 x,y : integer;
begin
deteksitepi();
smoothing();
 for y := 0 to image1.Height-1 do
 begin
  for x := 0 to image1.Width-1 do
  begin

  if (img1=true) and (img3=true) then
  begin
   if (bitmapHasilR[x,y] > 150) or
(bitmapHasilG[x,y] > 150) or
(bitmapHasilB[x,y] > 150) then
    begin
     Image4.Canvas.Pixels[x,y] :=
RGB(bitmapHasilR[x,y],
bitmapHasilG[x,y], bitmapHasilB[x,y]);
    end
    else
    begin
     Image4.Canvas.Pixels[x,y] :=
RGB(bitmapRB[x,y], bitmapGB[x,y],
bitmapBB[x,y]);
    end;
  end
  else if(img1=true) and (img2=true) then
  begin
   if (bitmapHasilR[x,y] > 150) or
(bitmapHasilG[x,y] > 150) or
(bitmapHasilB[x,y] > 150) then
    begin
     Image4.Canvas.Pixels[x,y] :=
RGB(bitmapTR[x,y], bitmapTG[x,y],
bitmapTB[x,y]);
    end
    else
    begin
     Image4.Canvas.Pixels[x,y] :=
RGB(bitmapRB[x,y], bitmapGB[x,y],
bitmapBB[x,y]);
    end;
  end
  else
```

```
 end;                                                                    end;
end;
```

Setelah gambar di deteksi tepi, gambar akan disatukan dengan gambar background ataupun tekstur, sehingga gambar menyatu menjadi satu layaknya satu gambar.

Penjelasan Code:

```
for y:=0 to image1.height-1 do
 begin
  for x:=0 to image1.width-1 do
  begin
```

Image di liat pada perulangan

```
 if (img1=true) and (img3=true) then
```

Jika gambar 1 dan 3 ngeload gambar maka fungsi if ini akan berjalan

```
  if (bitmapHasilR[x,y] > 150) or (bitmapHasilG[x,y] > 150) or (bitmapHasilB[x,y] > 150) then
   begin
    Image4.Canvas.Pixels[x,y] := RGB(bitmapHasilR[x,y], bitmapHasilG[x,y], bitmapHasilB[x,y]);
   end
   else
   begin
    Image4.Canvas.Pixels[x,y] := RGB(bitmapRB[x,y], bitmapGB[x,y], bitmapBB[x,y]);
   end;
```

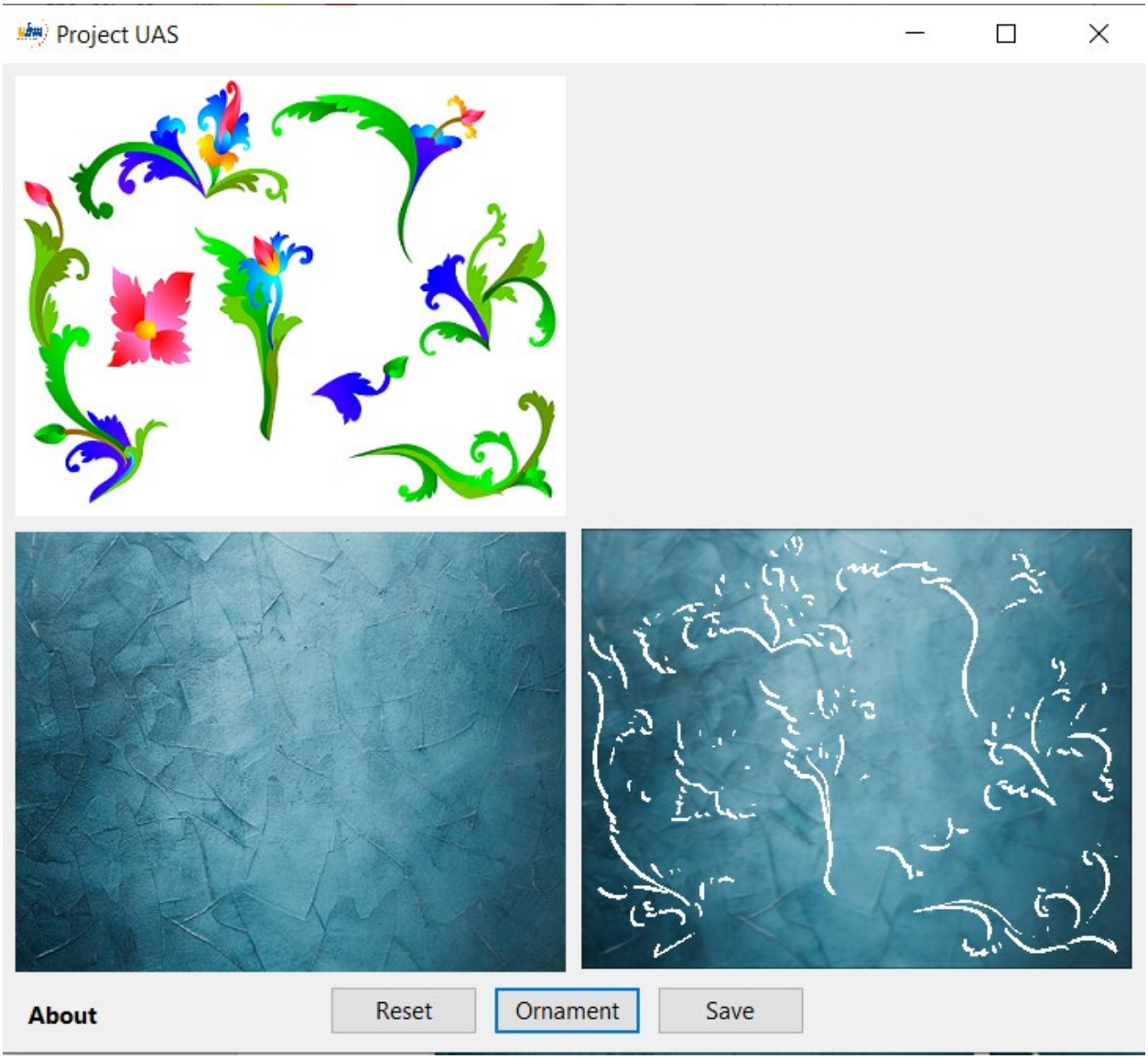
Jika hasil bitmap lebih besar dari 150, maka image 4 akan diisi dengan RGB(bitmapHasilR[x,y], bitmapHasilG[x,y], bitmapHasilB[x,y]) Yang telah dihitung di deteksi lalu jika <150 akan diisi RGB(bitmapRB[x,y], bitmapGB[x,y], bitmapBB[x,y])

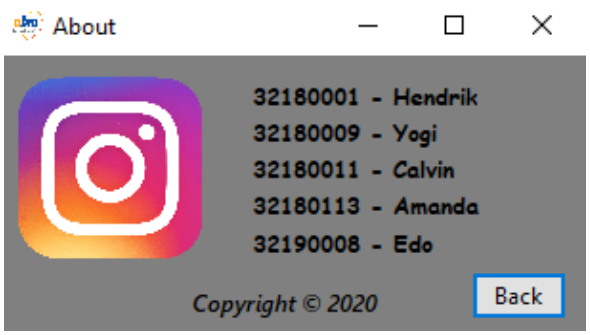
Jika gambar 1 dan 2 ngeload gambar maka fungsi if ini akan berjalan

```
else if(img1=true) and (img2=true) then
 begin
   if (bitmapHasilR[x,y] > 150) or (bitmapHasilG[x,y] > 150) or (bitmapHasilB[x,y] > 150) then
   begin
    Image4.Canvas.Pixels[x,y] := RGB(bitmapTR[x,y], bitmapTG[x,y], bitmapTB[x,y]);
   end
   else
   begin
    Image4.Canvas.Pixels[x,y] := RGB(bitmapRB[x,y], bitmapGB[x,y], bitmapBB[x,y]);
   end;
  end
```

Jika hasil bitmap lebih besar dari 150, maka image 4 akan diisi dengan RGB(bitmapTR[x,y], bitmapTG[x,y], bitmapTB[x,y]) Yang telah dihitung di deteksi lalu jika <150 akan diisi RGB(bitmapRB[x,y], bitmapGB[x,y], bitmapBB[x,y])

4.2. Hasil Implementasi





BAB V
PENUTUP

Kesimpulan

Dari yang kita tahu bahwa sebuah gambar memiliki nilai pixel masing-masing yang mana dapat diubah sesuai yang kita inginkan untuk mencari ketepatan atau kecocokan warna yang baik. Untuk menciptakan hal tersebut kita membutuhkan rumus-rumus atau *code* program yang benar agar dapat berjalan dengan baik dan tidak terjadi error. Aplikasi kami sudah berhasil dibuat dan sukses dalam menjalankan aplikasi yang kami buat dengan baik tanpa adanya error. Kami membuat aplikasi ini dengan menggunakan 2 bahasa program yaitu Bahasa pascal dan IDE Lazarus, pada aplikasi kami ini terdapat 4 button yang kami gunakan pada menu utama seperti Reset, Ornament, Save, dan About, dan ada 1 button yang kami program untuk memasukan Citra yang nantik akan kami gabungkan untuk menghasilkan sebuah Citra yang berupa Ornament dan pada 2 gambar tersebut akan kami gunakan utuk memproses Ornament, mereset canvas dan juga untuk melakukan save pada citra hasil yang sudah dihasilkan dari proses tadi. Pada aplikasi yang kami buat ini kami menggunakan metode dengan menggunakan gabungan dari beberpa proses seperti Grayscale, Filtering, Deteksi Tepi, Biner, Invers, dan Smoothing (pada proses ini kami menggunakannya pada bagian background) untuk menghasilakn sebuah ornament, aplikasi yang kami buat ini prosesnya dengan mengambil 2 gambar objek yang sudah kami masukan tadi pada citra dan kemudian akan di gabungkan dengan citra background dengan menggunakan proses Smoothing untuk background agar menghasilkan citra yang bagus dan mulus.

DAFTAR PUSTAKA

1. https://pemrogramanmatlab.com/2017/07/26/pengolahan-citra-digital/
2. https://pemrogramanmatlab.com/pengolahan-citra-digital/perbaikan-kualitas-citra/
3. http://seputar-programming-it.blogspot.com/2015/11/pengolahan-citra-4-brightness-invers.html