BAB 3

ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Analisis sistem adalah penguraian dari suatu sistem informasi yang utuh kedalam bagian-bagian komponennya dengan maksud untuk mengidentifikasikan dan mengevaluasi permasalahan, kesempatan, hambatan yang terjadi dan kebutuhan yang diharapkan sehingga dapat diusulkan perbaikan. Yogiyanto (1995).

Dalam membangun sebuah sistem, di perlukan sebuah analisis terlebih dahulu karena tanpa menganalisis sistem yang ada maka aplikasi tidak bisa di bangun. Dalam menganalisis sistem secara *Object Oriented* yang nantinya di bangun aplikasi berorientasi objek, di perlukan pendefinisian kasus dari sistem tersebut. Berikut adalah hasil analisis dari sistem manual yang ada dengan menggunakan metode UML (*Unified Modeling Language).*

3.1.1 Analisis Masalah

Penyimpanan citra dalam bentuk digital sekarang ini sudah hal yang sangat lazim. Dimana untuk saat ini hampir hasil pengambilan gambar disimpan dalam memori atau berbentuk digital. Akan tetapi, terkadang tidak semua citra digital dapat langsung digunakan oleh pengguna karena kualitas citra yang dihasilkan kurang baik dan perlu diolah terlebih dahulu sehingga siap untuk digunakan. Masalah yang biasa dihadapi adalah adanya derau atau noise yang mengakibatkan citra kotor atau cacat.

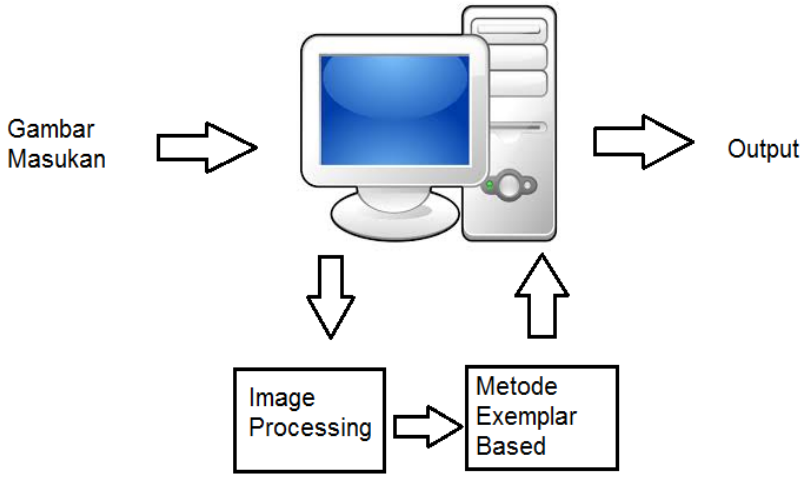
Teknik *inpainting* juga biasanya digunakan untuk merestorasi foto yang lama, biasanya mengalami kerusakan atau degradasi pada permukaannya. Kerusakan foto ini oleh tumbuhnya jamur atau tergores dikarenakan penyimpanan kurang baik. Untuk ini perlu dilakukan suatu proses rekonstruksi atau *recovery* sebuah citra. Sebelum dilakukan rekontruksi, citra harus diubah menjadi citra digital yang

sebelumnya adalah citra analog atau foto yang tercetak dikertas. Maka perlu dilakukan restorasi untuk mendapatkan suatu citra yang seperti aslinya kembali.

Berdasarkan uraian yang dikemukakan, maka terdapat masalah bagaimana cara membangun aplikasi *inpainting* untuk merestorasi gambar dan menghapus objek pada gambar digital.

3.1.2 Analisis Sistem Yang Akan Dibangun

Sistem yang akan dibangun adalah sistem untuk merestorasi citra atau mengembalikan citra yang sudah terkena noda atau derau seolah – olah kembali lagi seperti citra aslinya. Sistem bisa secara otomatis merestorasi citra tetapi sebelumnya menandai daerah yang akan direstorasi. Adapun gambaran umum tentang sistem yang akan dibangun dilihat pada gambar 3.1 :

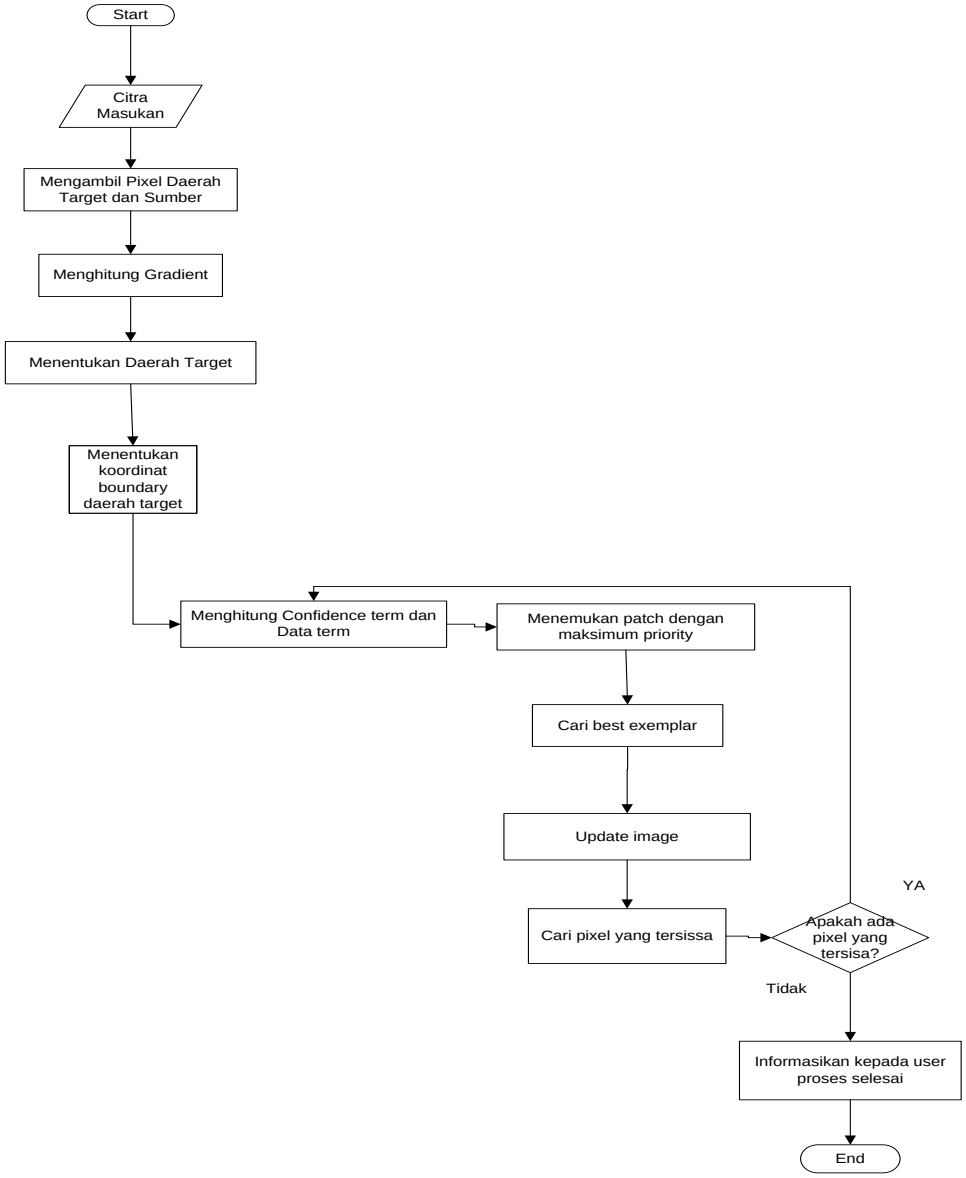


Gambar 3.1 Alur Proses Yang akan Dibangun

Gambar 3.1 menggambarkan proses secara umum yang akan dibangun. Dari menginputkan gambar yang akan direstorasi kemudian pengolahan citra dan penggunaan metode e*xemplar based*. Hasil dapat dilihat dilayar komputer.

3.1.3 Analisis Restorasi Citra

Pada bagian sebelumnya sudah dijelaskan proses restorasi citra yang akan dibangun secara umum. Berikut gambar 3.2 merupakan detail dari proses yang akan dibangun :



Gambar 3.2 *Flowchart* sistem yang akan dibangun

Berikut merupakan alur sistem yang akan dibangun :

1. Citra masukan, pengguna memilih citra yang akan direstorasi.
2. Mengambil piksel daerah target dan daerah sumber. Proses ini terjadi setelah user menandai daerah yang akan direstorasi.
3. Menghitung gradient daerah sumber dan daerah target.
4. Menentukan daerah target ditandai dengan warna hijau. Daerah target yang kemudian disebut *Region of Interest* (ROI) dihasilkan dengan cara memilih daerah mana yang akan dihapus dengan memberikan titik-titik penghubung. Daerah yang telah dibatasi dengan titik-titik tersebut kemudian akan diberi indikator warna hijau, hal ini menandakan bahwa daerah yang telah diberi warna indikator tersebut merupakan daerah target sedangkan daerah lainnya merupakan daerah sumber.
5. Menentukan koordinat *boundary* daerah target disini untuk menentukan tepi dari daerah target. Penentuan daerah tepi ini penting untuk proses pengisian daerah target.
6. Menghitung *confidence term* (informasi tekstur) dan *data term* (informasi *isophote* atau struktur). Kedua value ini untuk mendapatkan potongan terbaik.
7. Menemukan potongan dengan maksimum prioritas. Disini proses dimana menemukan potongan terbaik.
8. Cari *best exemplar* merupakan proses mencari potongan yang sesuai untuk mengisi daerah target.
9. *Update* citra merupakan proses mengganti *patch* atau potongan yang didapat dari penyisipan *exemplar*. Jadi patch yang sebelumnya masih menjadi daerah target akan berubah menjadi daerah sumber. Kemudian proses ini diulangi lagi sampai piksel yang berada didalam daerah target tersebut habis.

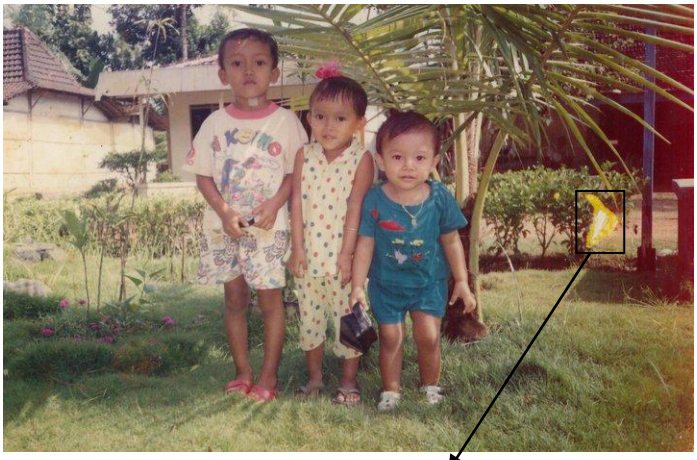
Penjelasan lengkap untuk setiap tahapan akan dijelaskan pada bagian berikut :

3.1.3.1 Menentukan Daerah Target

Proses ini dimana citra masukan akan dibagi menjadi dua daerah yaitu dareah target dan daerah sumber. Daerah target inilah yang akan diolah dan digantinya dengan piksel dari daerah sumber. Berikut proses dari penentuan daerah target dan daerah sumber :

1. Gambar masukan

   Gambar masukan ini merupakan gambar yang akan direstorasi. Pada gambar 3.3 ada derau yang dikarenakan penyimpanan yang kurang baik sewaktu citra masih berbentuk cetak sebelum dijadikan citra digital.

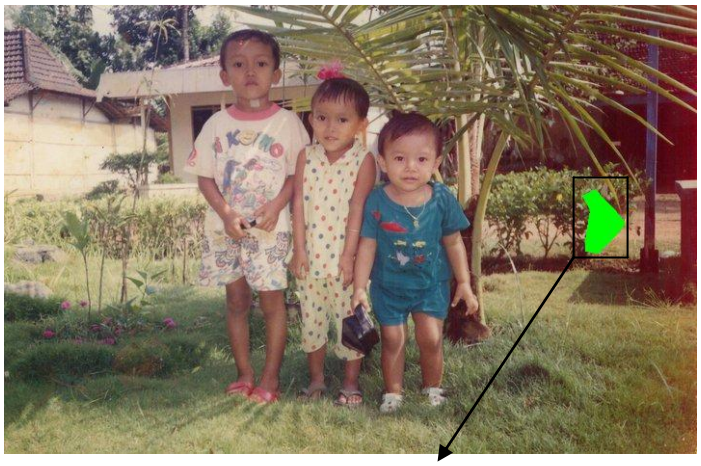


6782130	9282775	6849970	7443132	9220309	7641791
5729703	5138336	9350116	7640521	8299722	9747932
9349866	4020895	5073595	12048896	9288159	9420756
9943025	7903193	1653386	4943802	9814513	8828889
10141674	4615065	9305	1984132	2445715	4222386
10931192	3101572	601700	2247304	1720968	5274309
8561125	1455230	2113427	2377109	3233938	3497359
7441626	1651846	3428006	3955110	4416666	4351890
7375574	4678314	5073575	5863087	5139619	5139

Gambar 3.3 Gambar Masukan dan Matrik RGB dari daerah yang ada noise

2. Gambar yang sudah ditandai daerah targetnya dan nilai piksel daerah target dan daerah sumber.

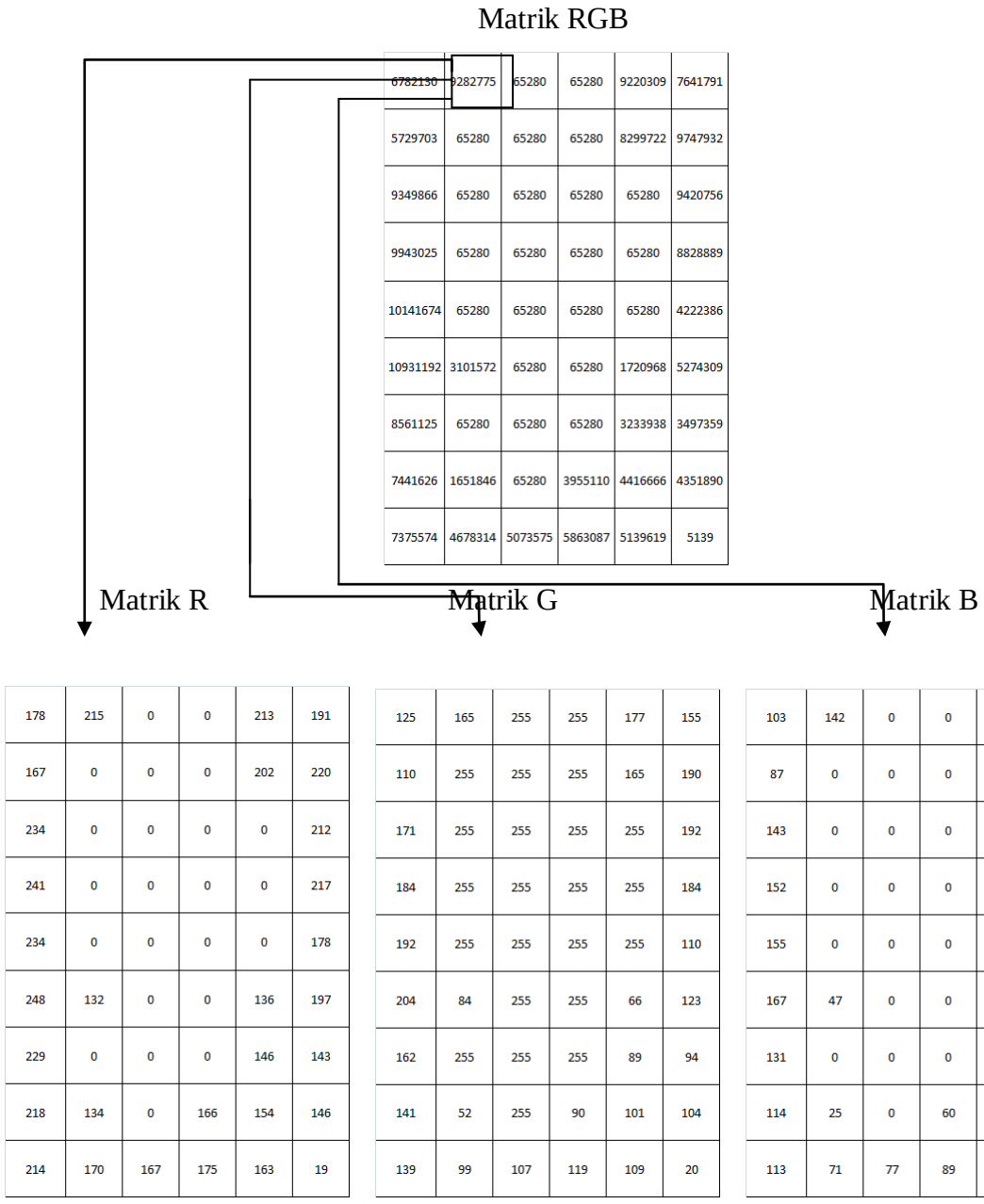
Gambar 3.4 merupakan gambar yang sudah ditandai sebagai daerah target dan daerah sumber, Dimana daerah target yang diwarnai warna hijau dan selain yang diwarnai warna hijau merupakan daerah sumber.



6782130	9282775	65280	65280	9220309	7641791
5729703	65280	65280	65280	8299722	9747932
9349866	65280	65280	65280	65280	9420756
9943025	65280	65280	65280	65280	8828889
10141674	65280	65280	65280	65280	4222386
10931192	3101572	65280	65280	1720968	5274309
8561125	65280	65280	65280	3233938	3497359
7441626	1651846	65280	3955110	4416666	4351890
7375574	4678314	5073575	5863087	5139619	5139

Gambar 3.4 Gambar yang sudah ditandai daerah target dan matrik RGB

Setelah mendapatkan nilai RGB tahap selanjutnya akan dikonversi ke masing – masing matrik R, matrik G, dan matrik B. Hasil konversi matrik dapat dilihat gambar 3.5 berikut.



Gambar 3.5 Hasil Konversi Matrik RGB menjadi Matrik R,G,B

Berikut merupakan algoritma dari proses menandai daerah target :

```
Procedure tandai(Graphics g)
```

```
Kamus
   lastX, lastY, firstX, firstY, currX, currY: integer
   entryImage: image
   PolygonCoordinatesX, PolygonCoordinatesY : vector

Algoritma

      lastX  ← -1
      lastY  ← -1
      firstX ← -1
      firstY ← -1
      currX  ← -1
      currY  ← -1

g.drawImage(entryImage, 0, 0, this)

{warna merah untuk poligon koordinat pertama }

if (firstX <> -1)
      g.setColor(Color.red)
      g.drawRect(firstX - 5, firstY - 5, 10, 10)
      g.drawRect(firstX - 4, firstY - 4, 8, 8)
endif

      g.setColor(Color.black);
      g.drawRect(0, 0, getWidth(), getHeight());
      g.setColor(Color.green);

if (PolygonCoordinatesX ←  null || PolygonCoordinatesY ←  null)
      return
endif

(*warna hitam untuk semua poligon koordinat kecuali koordinat
pertama*)
g.setColor(Color.black);
for i←1 to i<PolygonCoordinatesX.size() do
      g.drawRect((Integer)      PolygonCoordinatesX.get(i)      -
5(Integer)PolygonCoordinatesY.get(i) - 5, 10, 10);
```

```
      g.drawRect((Integer)       PolygonCoordinatesX.get(i)       -
4,(Integer) PolygonCoordinatesY.get(i) - 4, 8, 8);
endfor
      g.setColor(Color.green);

if (lastX <> -1)
{menyambungkan koordinat dengan garis hijau diantara titik
koordinat }
      g.drawLine(lastX, lastY, currX, currY)
endif
```

3.1.3.2 Menentukan Gradient

Penentuan gradient daerah target dan daerah sumber disini digunakan untuk mengetahui tepi dari daerah target yang nantinya akan digunakan untuk mendapatkan titik patch. Berikut merupakan algoritma dari penentuan gradient daerah target dan daerah :

```
Procedure calculateGradient(a[][],ih,iw)

Kamus
i,j : integer
gradientX, gradientY: array of double
row, column : array of integer
h : vector

Algoritma
            gradientX ←  new double[ih][iw]
            gradientY ←  new double[ih][iw]
            h ←  new Vector()
            row ←  new int [ih]
            column ←  new int [iw]

            for i ←  0 to i < ih do
                  row[i] ←  i + 1
            endfor
```

```
            For  i ←  0 to i < iw do
                  column[i] ←  i + 1
            endfor

            if (ih > 1) then
                  for j ←  0 to j < iw do
                  gradientY[0][j]  ←      (a[1][j]  -  a[0][j])  /
(row[1] - row[0])
                  gradientY[ih - 1][j] ←   (a[ih - 1][j] - a[ih -
2][j]) / (row[ih - 1] - row[ih - 2]);
                  endfor
endif



            if (ih > 2) then
                  for i ←  1 to i < ih – 1 do
                        for j ←  0 to j < iw do
                        gradientY[i][j] ←   (a[i + 1][j] - a[i -
1][j]) / (row[i + 1] - row[i - 1])
                        endfor
                  endfor
            endif

            if (iw > 1) then
                  for j ←  0 to j < ih do
                        gradientX[j][0] ← (a[j][1] - a[j][0]) /
(column[1] - column[0]);
                        gradientX[j][iw - 1] ←  (a[j][iw - 1] -
a[j][iw - 2]) / (column[iw - 1] - column[iw - 2]);
                  endfor
            endif

            if (iw > 2) then
                  for i ←  0 to i < ih; do
                        for j ←  1 to j < iw – 1 do
                        gradientX[i][j] ←  (a[i][j + 1] - a[i][j -
1]) / (column[j + 1] - column[j - 1]);
```

```
                endfor
            endfor
        endif
```

3.1.3.3 Proses menentukan potongan atau patch dengan maksimum priority

Proses ini merupakan cara mendapatkan nilai piksel disekitar titik *patch* dan dibentuk sebuah *patch* berukuran 3x3. *Pixelmap* merupakan matriks yang merepresentasikan *pixelmap* gambar, p adalah posisi pixel *patch* sekitar yang harus dihitung, dan N matriks yang berisi koordinat piksel yang membentuk *patch*. Berikut merupakan algoritmanya :

```
Function getpatch(pixelmap, p)

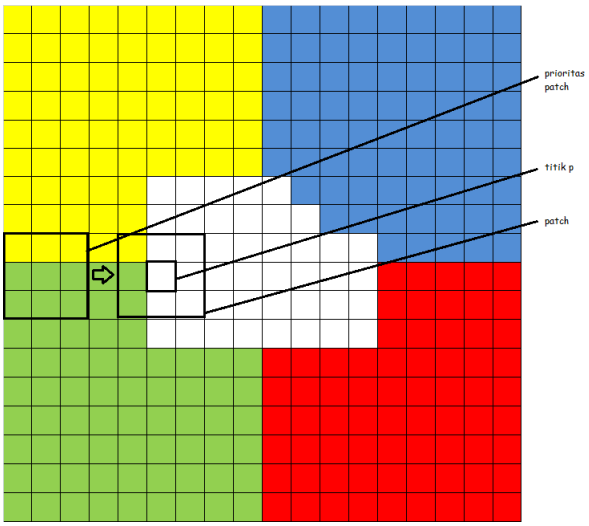
Kamus
   y,x,p,temp1.temp2,temp3,temp4,ih,iw,N : integer
Algoritma

        p ← p - 1;
        y ← p / ih;
        p ← p % ih;
        x ← p + 1;
        {Start koordinat X }
        temp1 ← max(x - w, 0)
        {End koordinat X }
        temp2 ← min(x + w, ih - 1)
        {Start koordinat Y }
        temp3 ← max(y - w, 0)
        {End koordinat Y }
        temp4 ← min(y + w, iw - 1)
   N[][]← new [temp4 - temp3 + 1][temp2 - temp1 + 1]

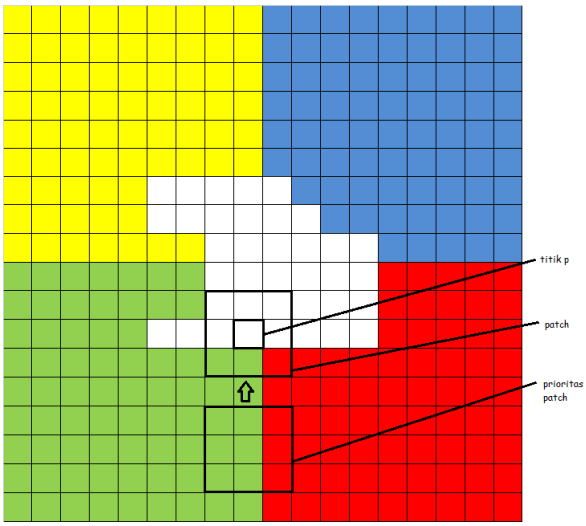
        for i ← 0 to i < temp4 - temp3 + 1 do
            for j ← 0 to j < temp2 - temp1 + 1 do
            N[i][j] ← temp1 + j + (temp3 + i) * ih
            endfor
        endfor
   return N
```

3.1.3.4 *Proses inpainting dengan metode* exemplar based

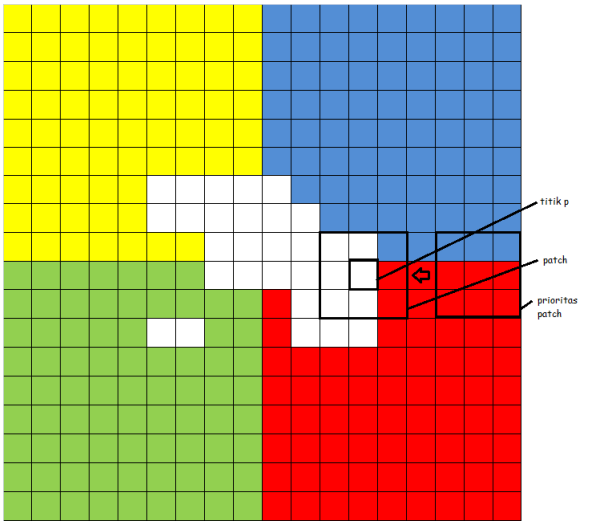
Sesudah didapatkan daerah target dan daerah sumber tahap selanjutnya proses *inpainting* degan metode exemplar based. Berikut contoh dari pengisian nilai pixel daerah target dari daerah sumber :



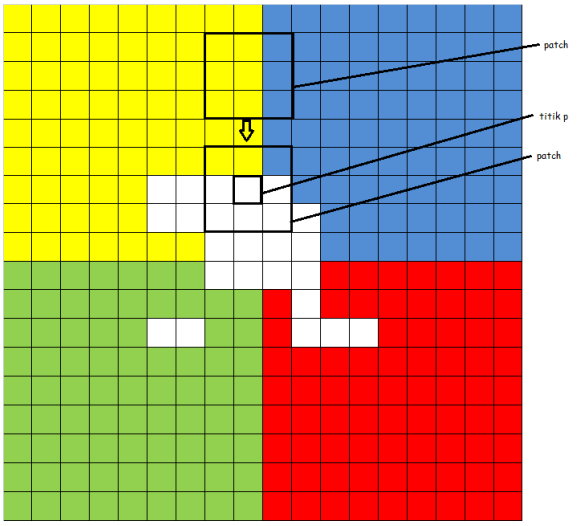
(i)



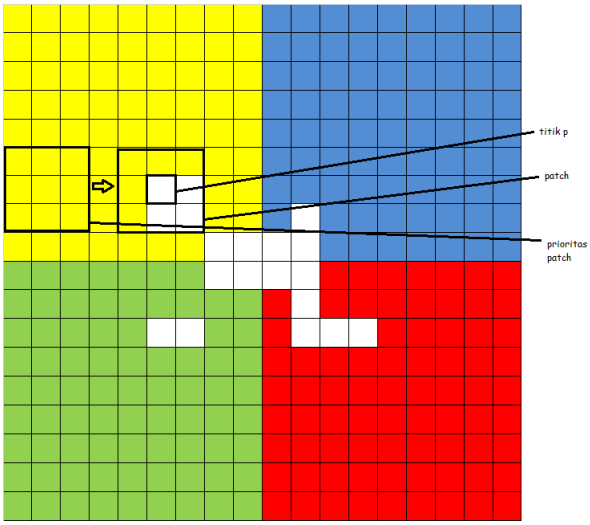
(ii)



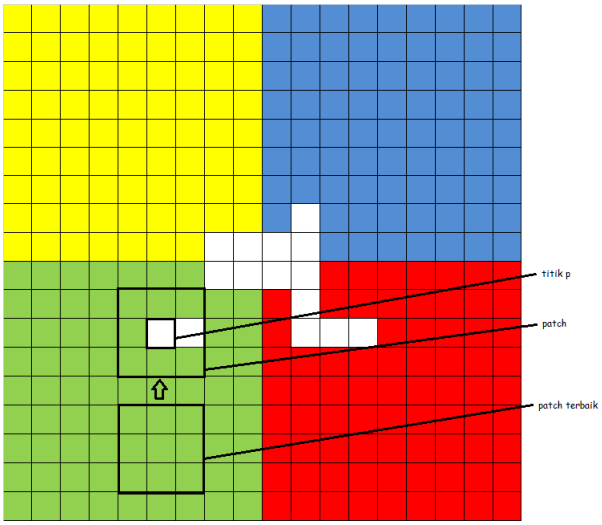
(iii)



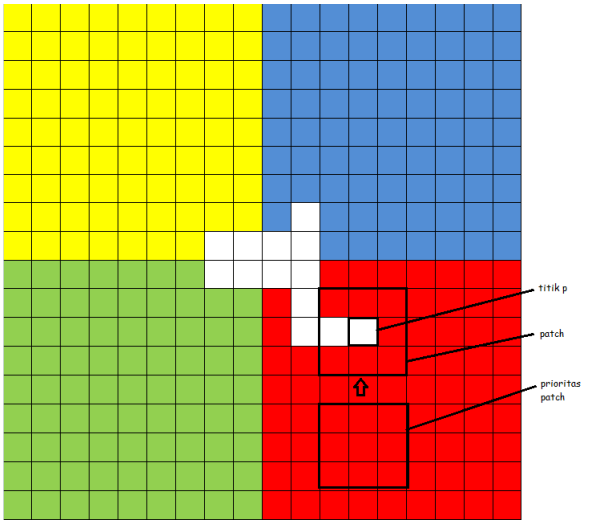
(iv)


titik p
patch
prioritas patch

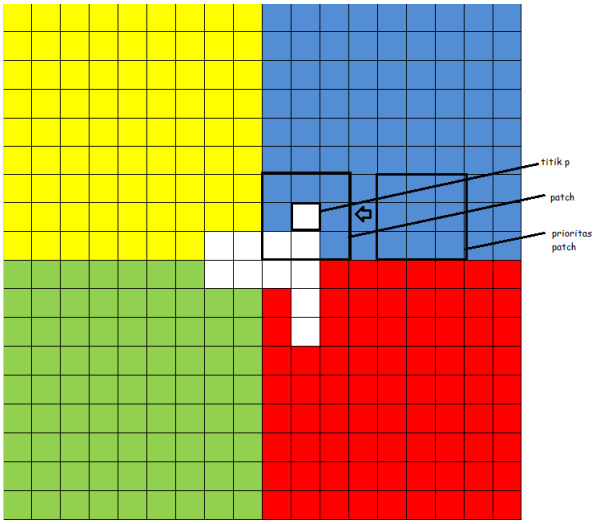
(v)


titik p
patch
patch terbaik

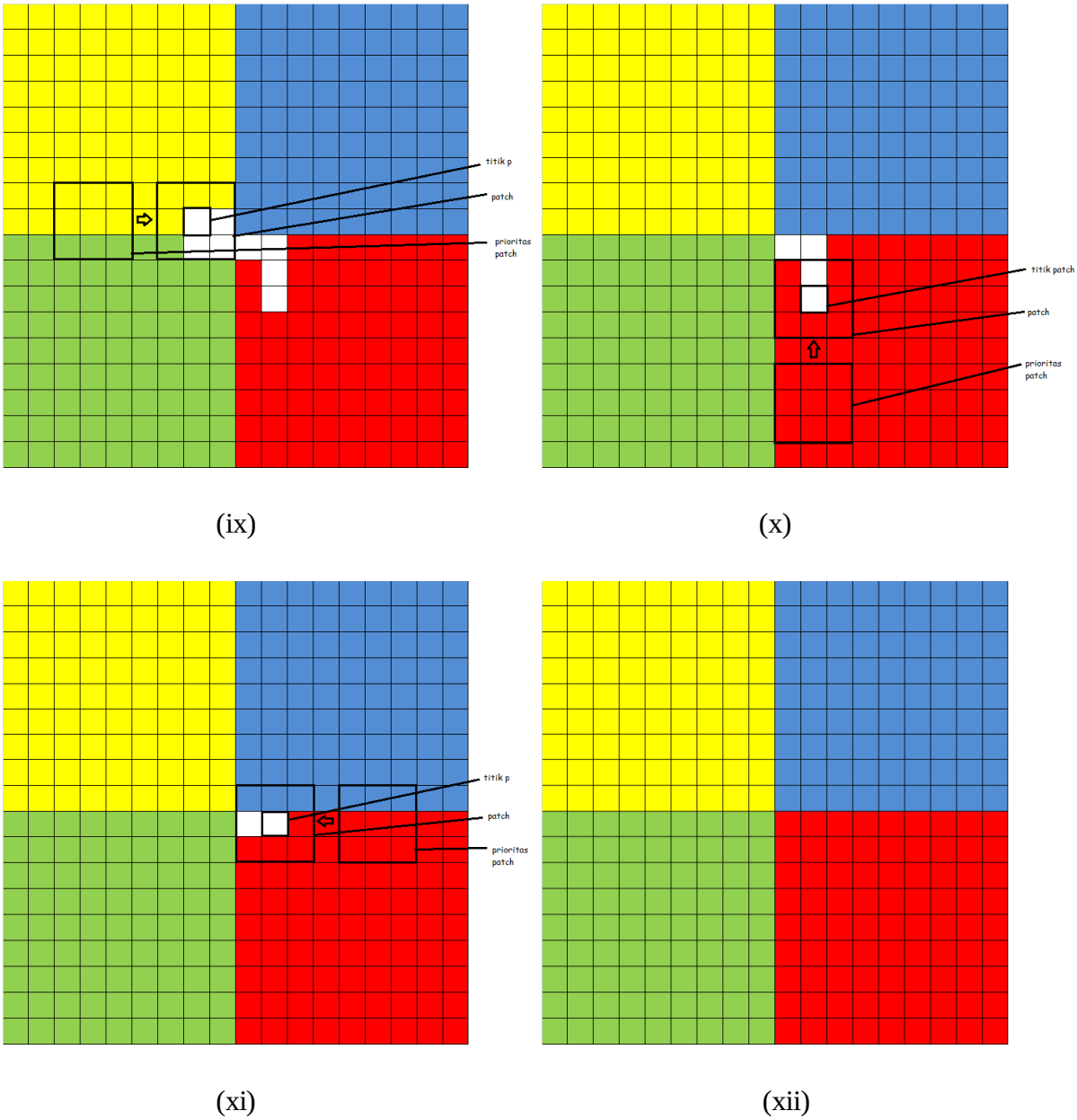
(vi)


titik p
patch
prioritas patch

(vii)


titik p
patch
prioritas patch

(viii)



(ix) (x)

(xi) (xii)

Gambar 3.6 Proses pengisian daerah target

Pada gambar 3.6 dijelaskan bagaimana proses pengisian daerah target. Pada gambar (i) merupakan gambar yang sudah ditandai daerah targetnya yaitu yang berwarna putih. Satu kotak dalam gambar itu diasumsikan satu piksel. Pada gambar (i) terdapat titik p yang merupakan acuan supaya didapatkan *patch*. *Patch* pada aplikasi ini menggunakan ukuran 3 x 3 piksel. Ukuran *patch* 3 x 3 piksel dipilih karena semakin kecil ukuran patch semakin bagus atau akurat dalam

memperoleh hasil restorasi. Ukuran patch disini dalam aplikasi ini ukuran yang paling kecil untuk dapat diterapkan.

Kemudian proses setelah itu mencari prioritas *patch* sesuai pada gambar (i). Setelah mendapatkan selanjutnya *patch* awal dari daerah target diganti *patch* baru dari daerah sumber sesuai gambar (ii). Arah dari pencarian patch ditentukan oleh tekstur (*Confidence term)* dan struktur (*Data term*) atau *ishopote*. Pada gambar (i) terdapat 2 tekstur warna yang berbeda kemudian dicari prioritas *patch* ke arah kiri sesuai arah tekstur dan strukturnya. Kemudian untuk piksel yang lainnya prosesnya seperti pengisian piksel pada gambar (i) sampai piksel didalam daerah target habis.

Berikut merupakan detail dari perhitungan *exemplar based* :

1. Prioritas *patch*
   a. Hitung *Confidence term C(p)* dengan persamaan sebagai berikut :

$$C(p) = \frac{\sum_{q \in \psi_p \cap \bar{\Omega}} C(q)}{|\psi p|} \qquad (3.1)$$

dimana:

$|\psi_p|$: luas ψ_p

Pada saat inisialisasi:

$$C(p) = 1 \leftrightarrow p \in \phi \text{ dan } C(p) = 0 \leftrightarrow p \in \Omega \qquad (3.2)$$

Algoritma inisialisasi *Confidence term C(p)* :

```
Procedure initialize_confidence_term()

Kamus
    confidence : array [ih][iw] of double
    ih,iw,i,j,p,r,g,b : integer
    pixelmap : array [][] of integer
Algoritma
          confidence ← new double[ih][iw]
          for i ← 0 to i < ih do
                for j ← 0 to j < iw do
                      p ← pixelmap[i][j]
                      r ← 0xff & p >> 16
```

```
                        g ← 0xff & p >> 8
                        b ← 0xff & p
                        if (r = 0 and g = 255 and b = 0)
                              confidence[i][j] ← 0
                        else
                              confidence[i][j] ← 1
                        endif
                  endfor
            endfor
```

b. Hitung *data term* dengan persamaan sebagai berikut

$$D(p) = \frac{|\nabla I \frac{1}{p} . np|}{\alpha} \quad (3.3)$$

dimana

α : factor normalisasi

$\nabla I \frac{1}{p}$: *isophote* (arah dan intensitas) pada titik *p*

Algoritma inisialisasi *data term* :

```
Procedure initialize_data_term()
Kamus
    i,j,ih,iw : integer
    data : array [ih][iw] of double
Algoritma
          data ← new double[ih][iw];
          for i ← 0 to i < ih do
                for j ← 0 to j < iw  do
                      data[i][j] ← -0.1;
                endfor
            endfor
```

2. Pencarian *Exemplar*

Pada proses ini menjelaskan cara memilih patch terbaik dan menggantikan daerah target dengan *best exemplar* yang telah diperoleh. Parameter hp merupakan *patch* dengan exemplar terbaik yang ditemukan, parameter toFill untuk piksel yang akan diisi *patch* dan parameter sourceRegion merupakan daerah

sumber untuk mencari exemplar terbaik. QuickInpaint sebagai tanda untuk melakukan proses *inpainting.* Berikut merupakan algoritma dari pencarian *exemplar* :

```
Function bestExemplar(Hp, toFill,sourceRegion,quickInpaint)
Kamus
i,j,col,row, int mm, nn, startX,startY, endX, endY, m ,n, ii, jj,
ii2, jj2, M,  N, I, J,best : integer
Ip,rIp,gIp,bIp, : array [][] of double
meanR, meanG, meanB : double
skipPatchFlag : boolean
Algoritma
Ip ← new double[toFill.length][toFill[0].length]
      for i ← 0 to i < toFill[0].length do
                  for j ← 0 to j < toFill.length do
                        col ← Hp[i][j] / ih
                        row ← Hp[i][j] % ih
                        Ip[j][i] ← pixelmap[row][col]
                  endfor
            endfor
            rIp ← new double [Ip.length][Ip[0].length]
            gIp ← new double [Ip.length][Ip[0].length]
            bIp ← new double[Ip.length][Ip[0].length]

            for i ← 0 to i < Ip.length do
                  for j ← 0 to j < Ip[0].length do

                        { Extract RGB komponent image }

      rIp[i][j] ← 0xff & Ip[i][j] >> 16
      gIp[i][j] ← 0xff & Ip[i][j] >> 8
      bIp[i][j] ← 0xff & Ip[i][j]
      endfor
      endfor
            m ← Ip.length
            n ← Ip[0].length
```

```
if (quickInpaint) then
 startX ← max(0, pixelPosX - n / 2 - continuousRow - diamX / 2)
 startY ← max(0, pixelPosY - m / 2 - continuousCol - diamY / 2)
 endX ← min(pixelmap[0].length - 1, pixelPosX + n / 2 +
continuousRow + diamX / 2)
 endY ← min(pixelmap.length - 1, pixelPosY + m / 2 + continuousCol
+ diamY / 2)
      mm ← endY - startY + 1
      nn ← endX - startX + 1

else
      mm ← pixelmap.length
      nn ← pixelmap[0].length
      startX ← 0
      startY ← 0
      endX ← 0
      endY ← 0
endif
      best ← { 0, 0, 0, 0 }
      skipPatchFlag ←false

            { untuk setiap patch }
            N ← startX + nn - n + 1
            M ← startY + mm - m + 1
for j ← startX to j < N do
  J ← j + n - 1;
      for i ← startY to i < M do
      I ← i + m - 1
      skipPatchFlag ← false

      meanR ← 0.0
      meanG ← 0.0
      meanB ← 0.0

            { menghitung patch error}
      for (jj = j, jj2 = 0; jj <= J; jj++, jj2++)
                  for (ii = i, ii2 = 0; ii <= I; ii++, ii2++)
```

```
       { Jika ada piksel tidak termasuk daerah sumber, lewati
patch.}

            if (sourceRegion[ii][jj] ≠ 1) then
                  skipPatchFlag ← true
                  break

            else if (toFill[ii2][jj2] = 0) then
                  rImage ← 0xff & pixelmap[ii][jj] >> 16
                  gImage ← 0xff & pixelmap[ii][jj] >> 8
                  bImage ← 0xff & pixelmap[ii][jj]

      { Hitung mean square error patch.}
                  err ← rImage - rIp[ii2][jj2]
                  patchErr ← err * err
                  err ← gImage - gIp[ii2][jj2]
                  patchErr ← err * err
                  err ← bImage - bIp[ii2][jj2]
                  patchErr ← err * err
{Hitung rata-rata nilai warna.ketika mendapatkan dua patch dengan
nilai error sama}
                  meanR ← rImage
                      meanG ← gImage
                      meanB ← bImage
                  endif
            endif
endfor
endfor
```

3. Update citra dan mencari piksel yang tersisa di daerah target

Update citra merupakan proses mengganti *patch* atau potongan yang didapat dari penyisipan *exemplar*. Jadi patch yang sebelumnya masih menjadi daerah target akan berubah menjadi daerah sumber. Kemudian proses ini diulangi lagi sampai piksel yang berada didalam daerah target tersebut habis. Berikut merupakan algoritma untuk mengudate citra :

```
Procedure     UpdateImage(pixelmap,fillregion,sourceregion,tofill,
confidence, gradientX, gradientY)
Kamus
I,j, ih,iw :  integer
Col,row,col1, row1 ,Confdence, gradientX,gradientY : array [][] of
integer
Algoritma
for i ← 0 to i < toFill.length do

   for j ← 0 to j < toFill[0].length do
      if (toFill[i][j] ≠ 0) then
          toFill[i][j] ← 1
          col ← Hp[i][j] / ih
          row ← Hp[i][j] % ih

          col1 ← Hq[i][j] / ih
          row1 ← Hq[i][j] % ih
{Rekontruksi daerah target,jika sudah dihapus dari fillregion}
          fillRegion[row][col] ← 0
{Rekontruksi daerah target, kemudian tambahkan ke SourceRegion}
           sourceRegion[row][col] ← 1;

{Propagate Confidence dan Isophote Values }
           confidence[row][col]  ←  confidence[p  %  ih][p  /  ih]
                     gradientX[row][col] ← gradientX[row1][col1]
          gradientY[row][col] ← gradientY[row1][col1]
           pixelmap[row][col] ← pixelmap[row1][col1]
{Update image pixels}

            color ← new integer [3]
                  color[0] ← 0xff & pixelmap[row1][col1] >> 16
                  color[1] ← 0xff & pixelmap[row1][col1] >> 8
                        color[2] ← 0xff & pixelmap[row1][col1]
                              raster.setPixel(col, row, color)

                              endif

                        endfor
```

```
      endfor
{Mencari pixel yang tersisa untuk dilakukan inpainting apakah
masih ada}

                  for i ←  0 to i < fillRegion.length do
                        for j ← 0  to j < fillRegion[0].length do
                              if (fillRegion[i][j] = 1) then
                                    flag = true

                              endif

                        endfor
                        endfor
```

Setelah semua piksel didalam daerah taget sudah diproses oleh sistem akan dihasilkan gambar 3.7 sebagai berikut :



Gambar 3.7 Citra Hasil

3.2 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional adalah kebutuhan yang tidak secara langsung terkait / berhubungan dengan fitur tertentu pada sistem. Kebutuhan non fungsional adalah meliputi kebutuhan perangkat yang diperlukan untuk membangun sistem baik perangkat keras maupun perangkat lunak.

3.2.1 Analisis Kebutuhan Perangkat Keras

Analisis kebutuhan perangkat keras menjelaskan tentang spesifikasi perangkat keras yang dibutuhkan untuk membangun dan menjalankan aplikasi yang akan dibangun.

Spesifikasi perangkat keras yang digunakan untuk membangun aplikasi ini adalah sebagai berikut :

1. Intel(R) Atom(TM) CPU N570 @1,66GHz
2. VGA 512 MB
3. RAM 1GB
4. Keyboard dan Mouse standar

3.2.2 Analisis Kebutuhan Perangkat Lunak

Analisis kebutuhan perangkat lunak menjelaskan tentang spesifikasi perangkat lunak yang dibutuhkan untuk membangun dan menjalankan aplikasi yang akan dibangun.

Perangkat lunak yang dibutuhkan untuk membangun aplikasi ini adalah sebagai berikut :

1. Sistem Operasi Microsoft Windows 7
2. Eclipse IDE

3.3 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional menyangkut analisis sistem yang dilakukan yaitu dengan menggunakan UML. Adapun tahapan analisis sistem menggunakan UML adalah melalui *use case diagram, Activity diagram, sequence diagram, dan class diagram*.

3.3.1 Analisis Kebutuhan Sistem

Analisi kebutuhan sistem ditujukan untuk memenuhi kebutuhan sistem yang akan dibangun. Adapun kebutuhan sistem yang diperlukan secara garis besar antara lain :

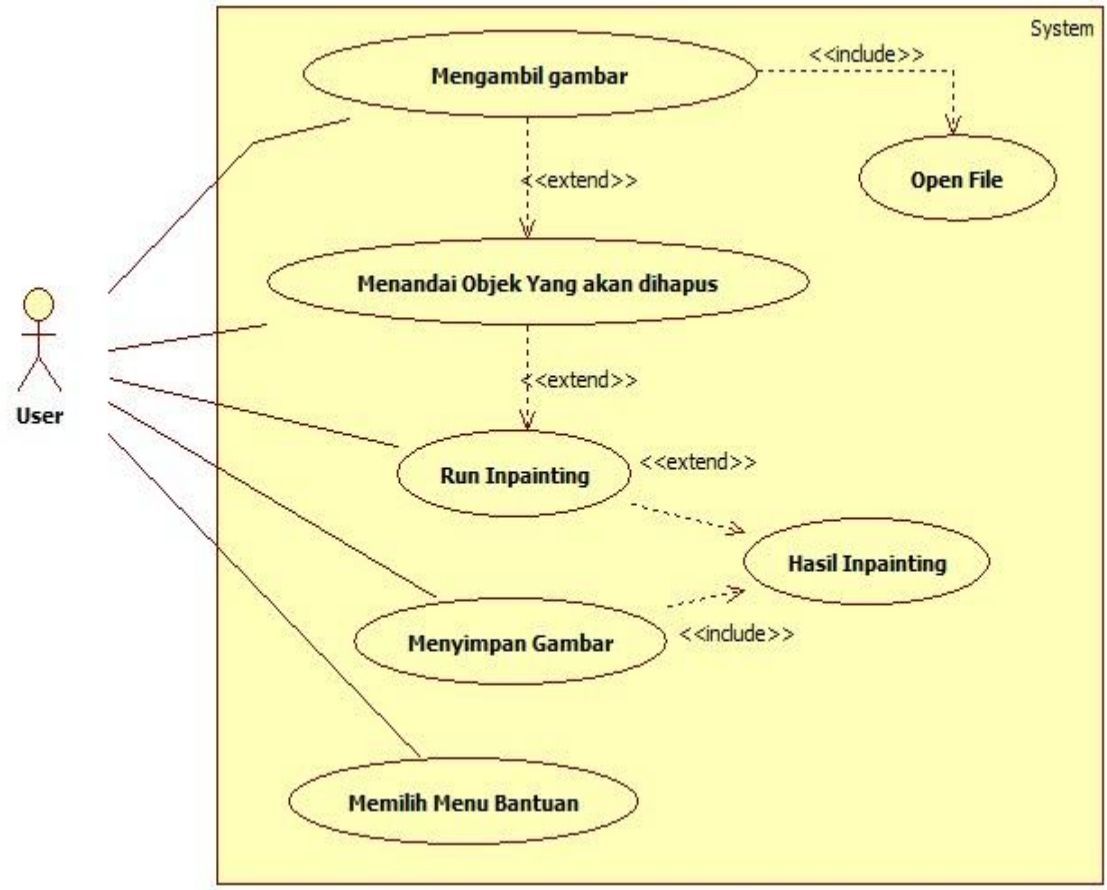
1. *User* menginputkan gambar ke dalam sistem.
2. *User* menandai objek yang akan dihapus yang disebut sebagai ROI (*Region Of Interest*) atau sebagai daerah target.
3. Setelah *user* menandai objek pada gambar, sistem akan melakukan proses *inpainting*. Dimana *pixel* daerah target itu akan diisi *pixel* dari daerah sumber.
4. Setelah *pixel* daerah target sudah tertutup semua maka proses *inpainting* selesai dan gambar hasil *inpainting* didapatkan.
5. *User* dapat menyimpan hasil gambar hasil *inpainting*.
6. *User* dapat memilih menu bantuan jika belum paham dengan cara kerja sistem *inpainting* ini.

3.3.2 Pemodelan Sistem

Pada tahap ini dilakukan pemodelan sistem untuk menentukan fungsi-fungsi yang dapat dilakukan oleh sistem aplikasi serta menentukan kelas yang dibutuhkan untuk realisasi fungsi-fungsi sistem yang telah dianalisis sebelumnya dan mendeskripsikannya kedalam bentuk diagram.

3.3.2.1 *Use case* diagram

Use case diagram merupakan diagram UML untuk menganalisa kebutuhan dan perilaku sistem. *Use case* juga bisa diartikan sebagai interaksi antara *user* dan sebuah sistem atau sub-sistem Secara grafik model kemampuan sistem disebut *use case*, sedangkan *user* disebut aktor dan mereka saling berinteraksi satu sama lain. Berikut merupakan *use case diagram* dari sistem yang dibangun :



Gambar 3.8 *Use case* Diagram

3.3.2.2 Definisi Aktor

Definisi aktor berfungsi untuk menjelaskan aktor yang terdapat pada *use case* diagram. Definisi aktor *pada use case* sistem yang dibangun dapat dilihat pada tabel berikut :

Tabel 3.1 Definisi Aktor

Aktor	Deskripsi
User	Merupakan aktor yang menggunakan aplikasi yang dibangun ini.

3.3.2.3 Definisi *Use case*

Definis *use case* merupakan penjelasan fungsi dari *use case diagram*. Berikut merupakan penjelasan definis *use case* sistem ini :

Tabel 3.2 Definisi *Use case*

No	*Use case*	Deskripsi
1	Mengambil gambar dair galeri	Merupakan fungsional untuk mengambil gambar pada folder galeri.
2	Menandai objek yang akan dihapus	Merupakan fungsional untuk menandai objek yang akan dihapus setelah mengambil gambar dari galeri.
3	Run *Inpainting*	Merupakan fungsional untuk menjalankan proses penghapusan dari objek yang sudah ditandai.
4	Hasil *Inpainting*	Merupakan fungsional hasil dari proses *inpainting.*
5	Memilih Menu Bantuan	Merupakan menu untuk menampilkan cara pemakaian aplikasi.
6	Hasil *Inpainting*	Merupakan fungsionaltias hasil dari *inpainting*.
7	Menyimpan Gambar	Merupakan fungsionalitas untuk menyimpan gambar hasil proses *inpainting*.

3.3.2.4 *Skenario* Use case

Skenario pada *use case* ini berisi alur dari *use case* utama dengan menjelaskan urutan interaksi antara aktor dan *use case* tersebut dari awal sampai akhir. Skenario proses-proses yang terdapat di dalam *Use case* dapat dilihat pada tabel-tabel dibawah ini :

Tabel 3.3 Skenario *use case* Mengambil gambar dari galeri

Identifikasi		
Nama	Mengambil gambar dari galeri	
Tujuan	Pengambilan gambar dari folder yang diinginkan *user*.	
Deskripsi	Fungsional untuk mengambil gambar yang ingin diproses.	
Aktor	*User*	
Skenario Utama		
Kondisi Awal	Menampilkan menu utama	
Aksi Aktor		Reaksi Sistem
1. Memilih menu file 2. Memilih sub menu *open image* 4. Melakukan pemilihan gambar		3. Menampilkan folder *image* 5. Menerima file yang diupload user, jika extensi tidak sesuai maka user akan memulai lagi pada proses awal.
Kondisi Akhir	Sistem menampilkan hasil pengambilan gambar dari galeri	

Tabel 3.4 Skenario *use case* Menandai Objek yang akan dihapus

Identifikasi	
Nama	Menandai Objek yang akan dihapus

Tujuan	Untuk mendapatkan ROI (*Region Of Interest*)	
Deskripsi	Fungsional untuk mendapatkan ROI atau objek yang akan dihapus.	
Aktor	*User*	
Skenario Utama		
Kondisi Awal	Menampilkan gambar yang akan di proses.	
Aksi Aktor		Reaksi Sistem
1. Menandai objek dengan cara meng- *click* bagian luar objek sampai tersambung lagi ke titik awal.		2 Memberi warna hijau objek yang sudah ditandai tadi sebagai ROI.
Kondisi Akhir	Sistem menampilkan hasil gambar yang sudah ditandai.	

Tabel 3.5 Skenario *use case Run Inpainting*

Identifikasi		
Nama	*Run Inpainting*	
Tujuan	Memproses gambar seusai objek yang akan dihapus sudah ditandai.	
Deskripsi	Fungsional untuk memproses gambar seusai objek yang akan dihapus sudah ditandai dan menjadi gambar baru.	
Aktor	*User*	
Skenario Utama		
Kondisi Awal	Menampilkan gambar yang akan di proses.	
Aksi Aktor		Reaksi Sistem
1. Memilih Menu *Inpainting* 2. Memilih sub menu *Run Inpainting*		3. Menampilkan proses *inpainting*
Kondisi Akhir	Sistem menampilkan hasil gambar setelah diproses.	

Tabel 3.6 Skenario *use case* Memilih Menu bantuan

Identifikasi		
Nama	Memilih Menu bantuan	
Tujuan	Memberikan informasi penggunaan aplikasi	
Deskripsi	Fungsional memberikan informasi penggunaan aplikasi kepada *user*.	
Aktor	*User*	
Skenario Utama		
Kondisi Awal	Menampilkan menu utama	
Aksi Aktor		Reaksi Sistem
1. Memilih menu Help 2. Memilih sub menu Bantuan		3. Menampilkan halaman bantu.
Kondisi Akhir	Sistem menampilkan halaman bantu yang berisi informasi penggunaaan aplikasi.	

Tabel 3.7 Skenario *use case* Hasil *Inpainting*

Identifikasi		
Nama	Hasil *Inpainting*	
Tujuan	Hasil dari penghapusan obyek dengan *inpainting*	
Deskripsi	Fungsional hasil dari penghapusan obyek dengan *inpainting*.	
Aktor	*User*	
Skenario Utama		
Kondisi Awal	Menampilkan gambar dengan obyek yang akan dihapus sudah ditandai.	
Aksi Aktor		Reaksi Sistem
		1. Menampilkan proses *inpainting* 2. Menampilkan alert jika proses sudah selesai.

	3. Menampilkan hasil dari *inpainting*
Kondisi Akhir	Sistem menampilkan hasil Inapinting.

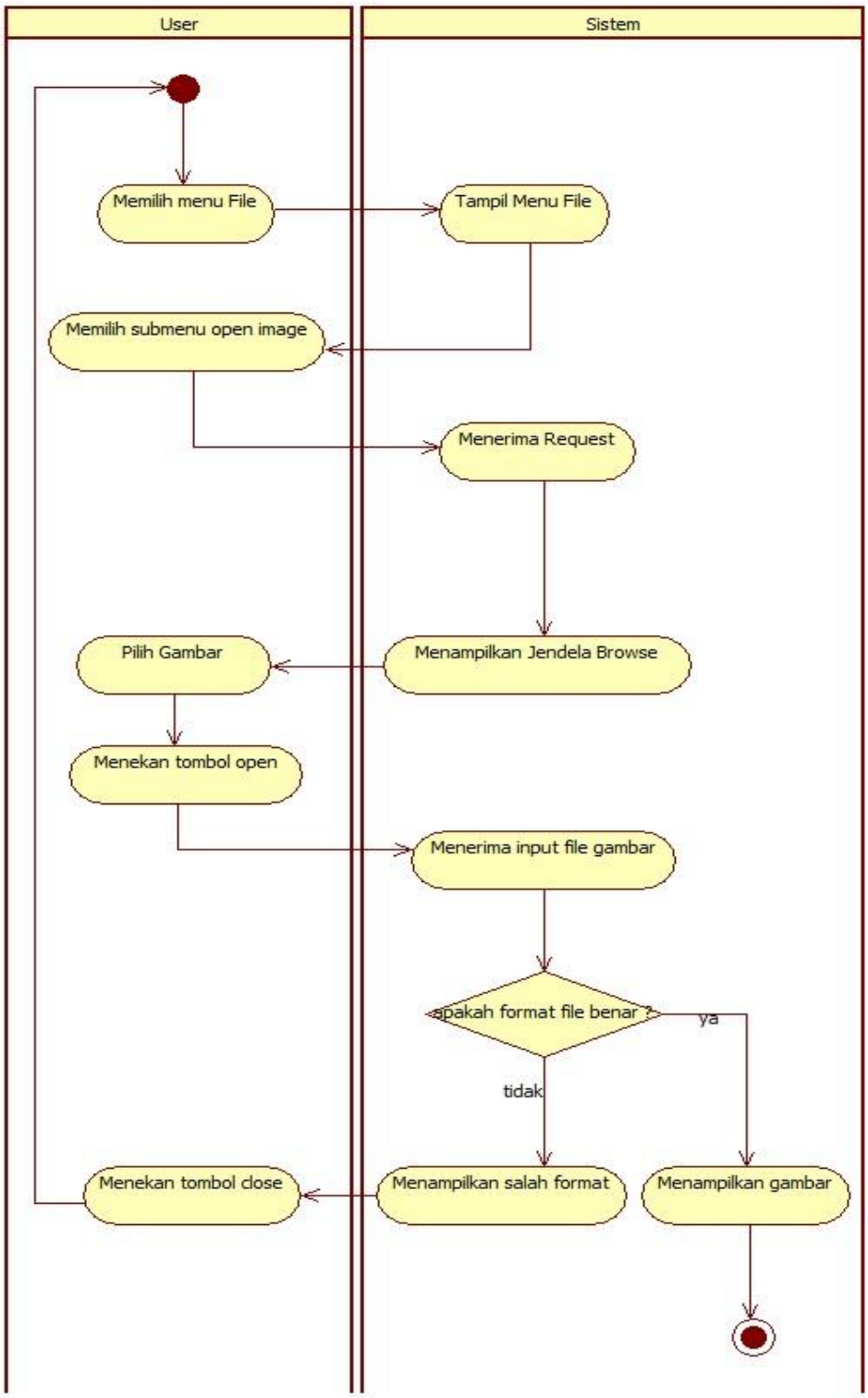
Tabel 3.8 Skenario *use case* Menyimpan Gambar

Identifikasi	
Nama	Menyimpan Gambar
Tujuan	Simpan file gambar hasil *inpainting*.
Deskripsi	Fungsional menyimpan hasil dari penghapusan obyek dengan *inpainting*.
Aktor	*User*
Skenario Utama	
Kondisi Awal	Menampilkan hasil *inpainting*.
Aksi Aktor	Reaksi Sistem
1. Memilih menu file 2. Memilih submenu *save* atau *save as* 4. Memilih *directory file* untuk menyimpan gambar.	3.Menampilkan *directory file*
Kondisi Akhir	Sistem menampilkan hasil Inapinting.

3.3.2.5 *Activity* Diagram

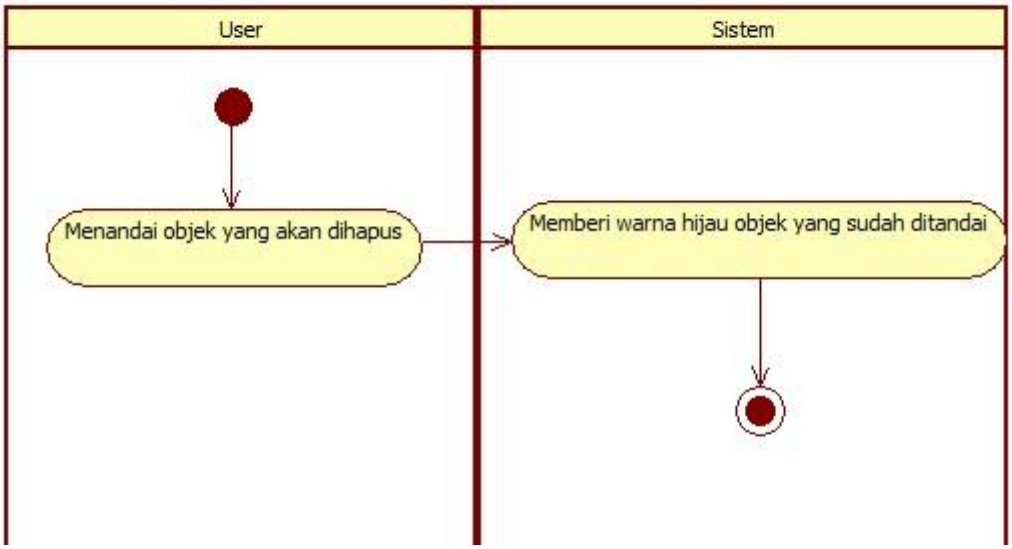
Activity Diagram merupakan bagian dari penggambaran sistem secara fungsional menjelaskan proses-proses logika atau fungsi yang terimplementasi oleh kode program. Aktivitas pertama selalu dimulai dari pengguna, kemudian akan mendapat respon dari sistem yang nantinya akan menampilkan akhir dari proses tersebut.

1. *Activity* Diagram Mengambil gambar



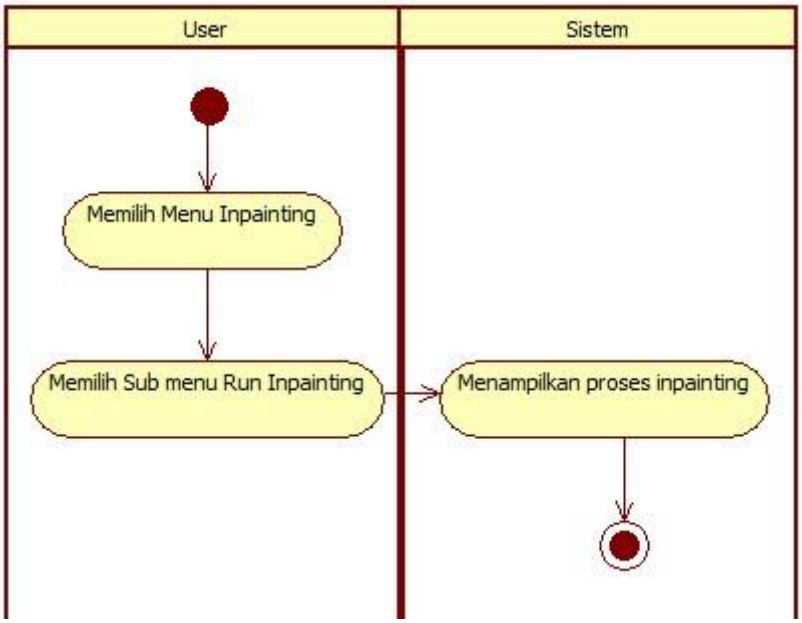
Gambar 3.9 *Activity* Diagram Mengambil gambar

2. *Activity* Diagram Menandai objek yang akan dihapus



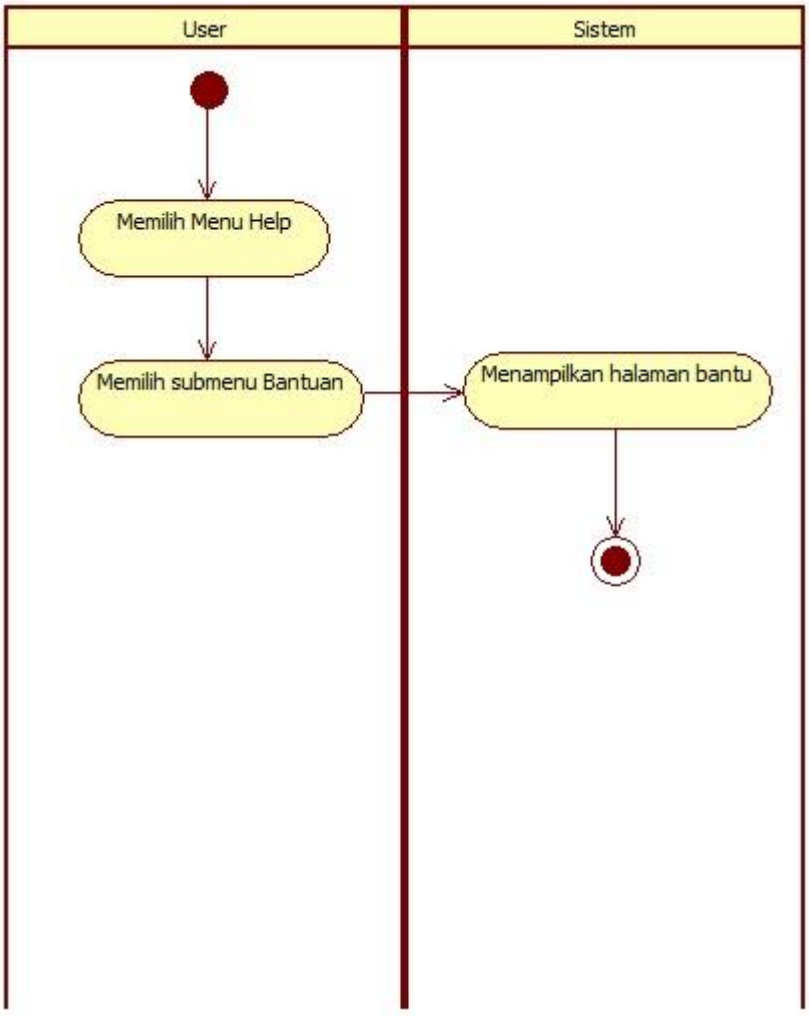
Gambar 3.10 *Activity* Diagram Menandai objek yang akan dihapus

3. *Activity* Diagram *Run Inapinting*



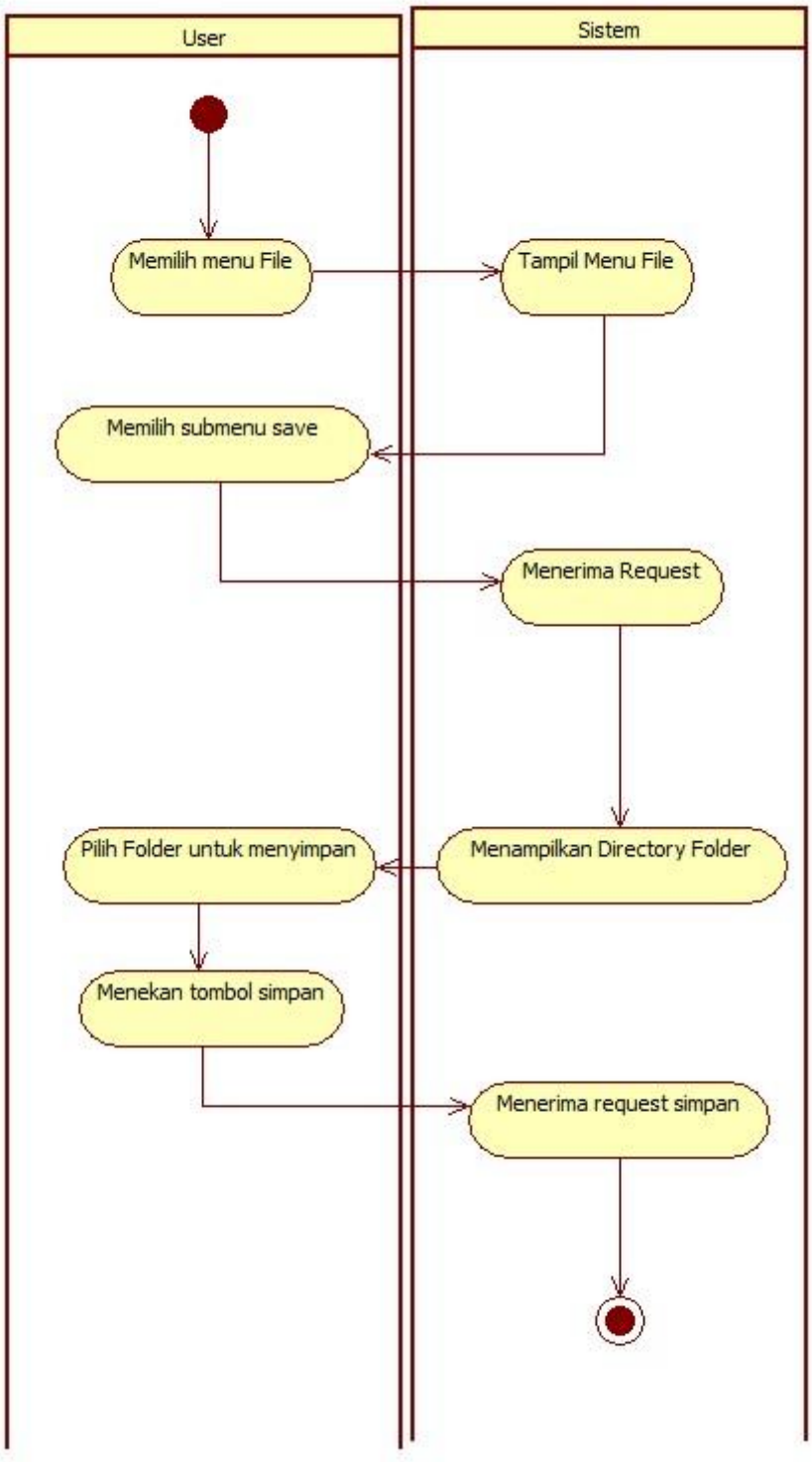
Gambar 3.11 *Activity* Diagram Run *Inpainting*

4. *Activity* Diagram Menu Bantuan



Gambar 3.12 *Activity* Diagram Memilih Menu Bantuan

5. *Activity* Diagram Menyimpan Gambar

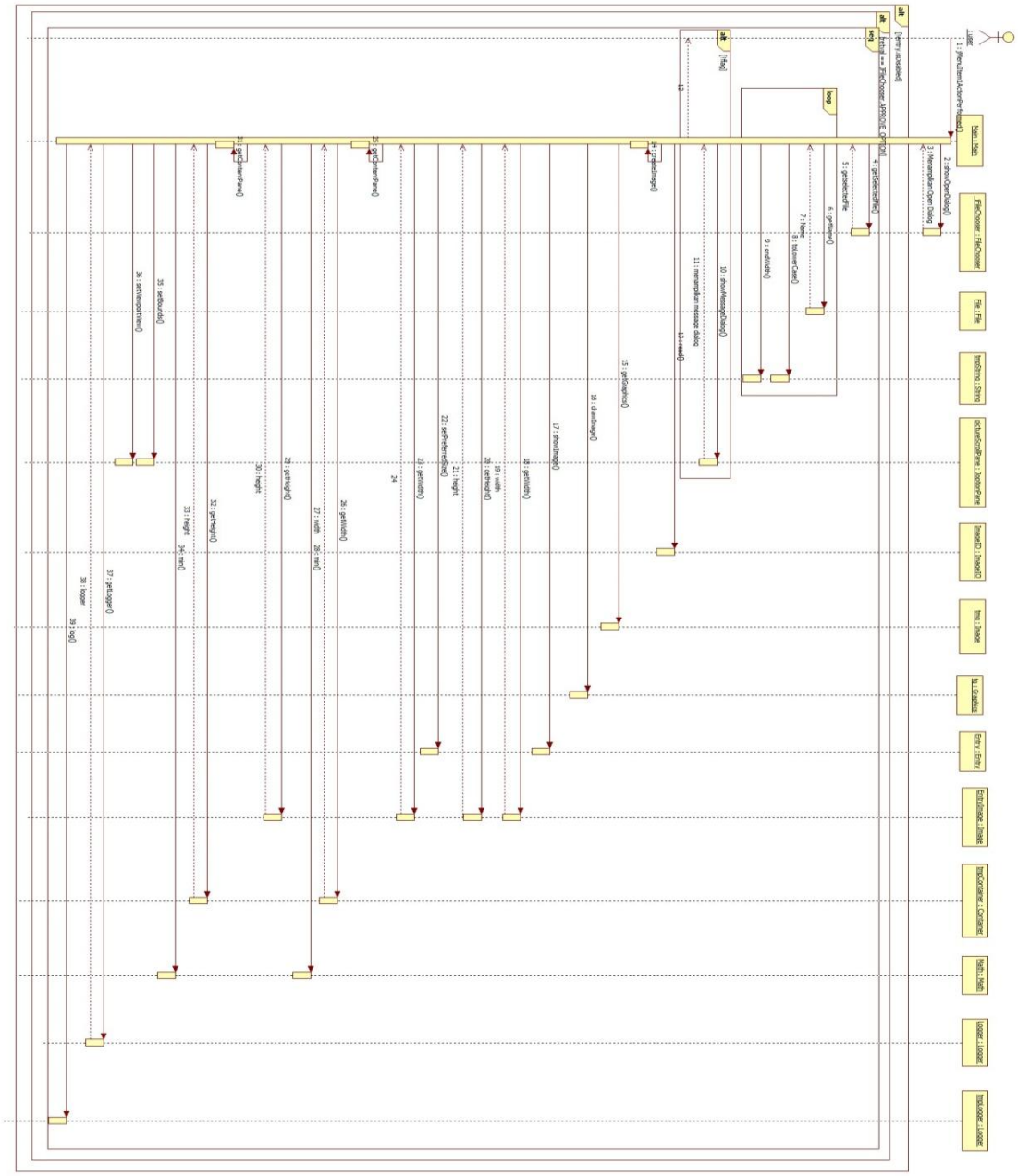


Gambar 3.13 *Activity* Diagram Menyimpan Gambar

3.3.3 *Sequence* Diagram

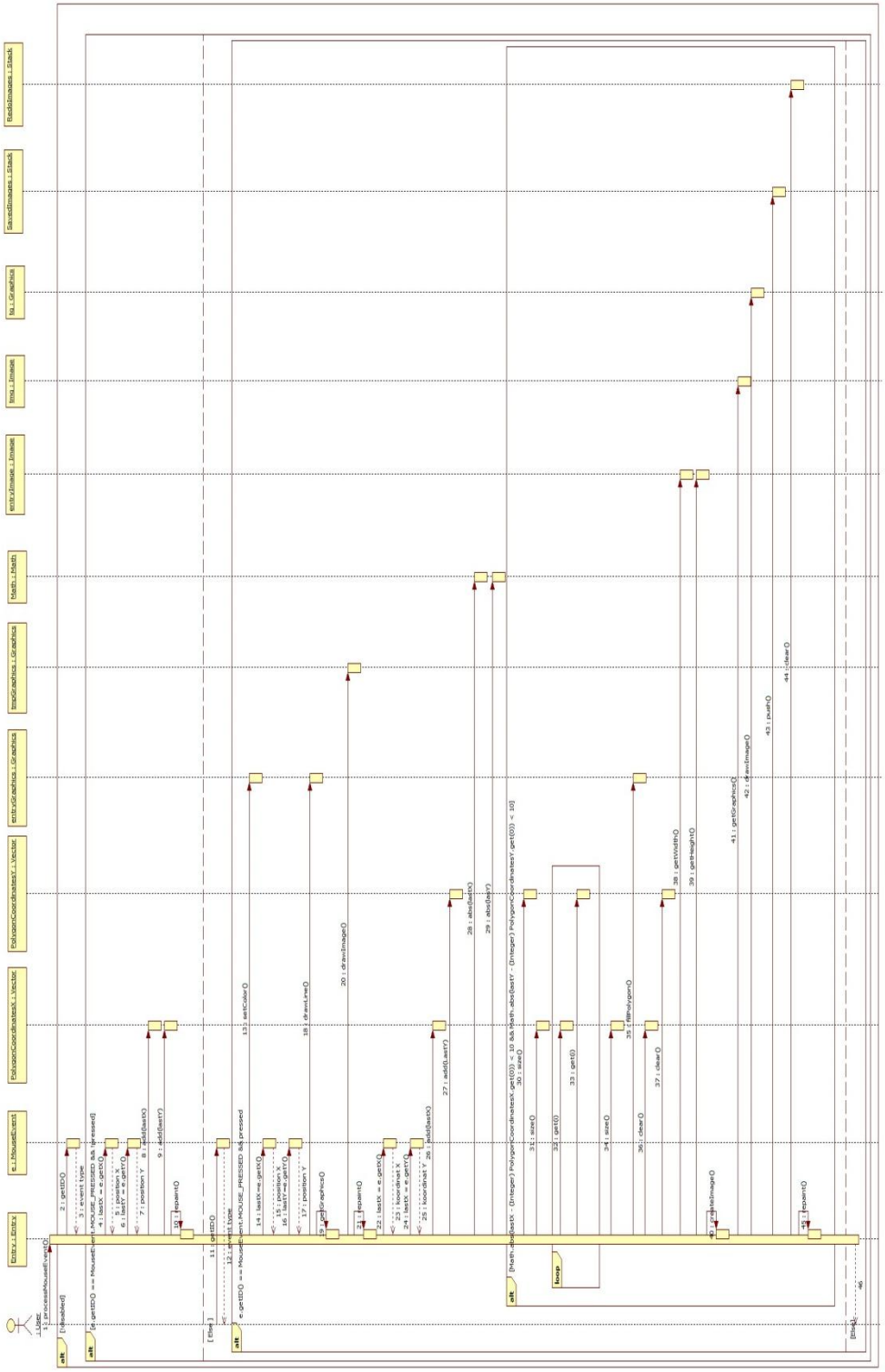
Pada tahap ini *Sequence Diagram* menjelaskan secara detail urutan proses yang dilakukan sistem untuk mencapai tujuan dari *Use case*. Interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.

1. *Sequence Diagram* Mengambil Gambar



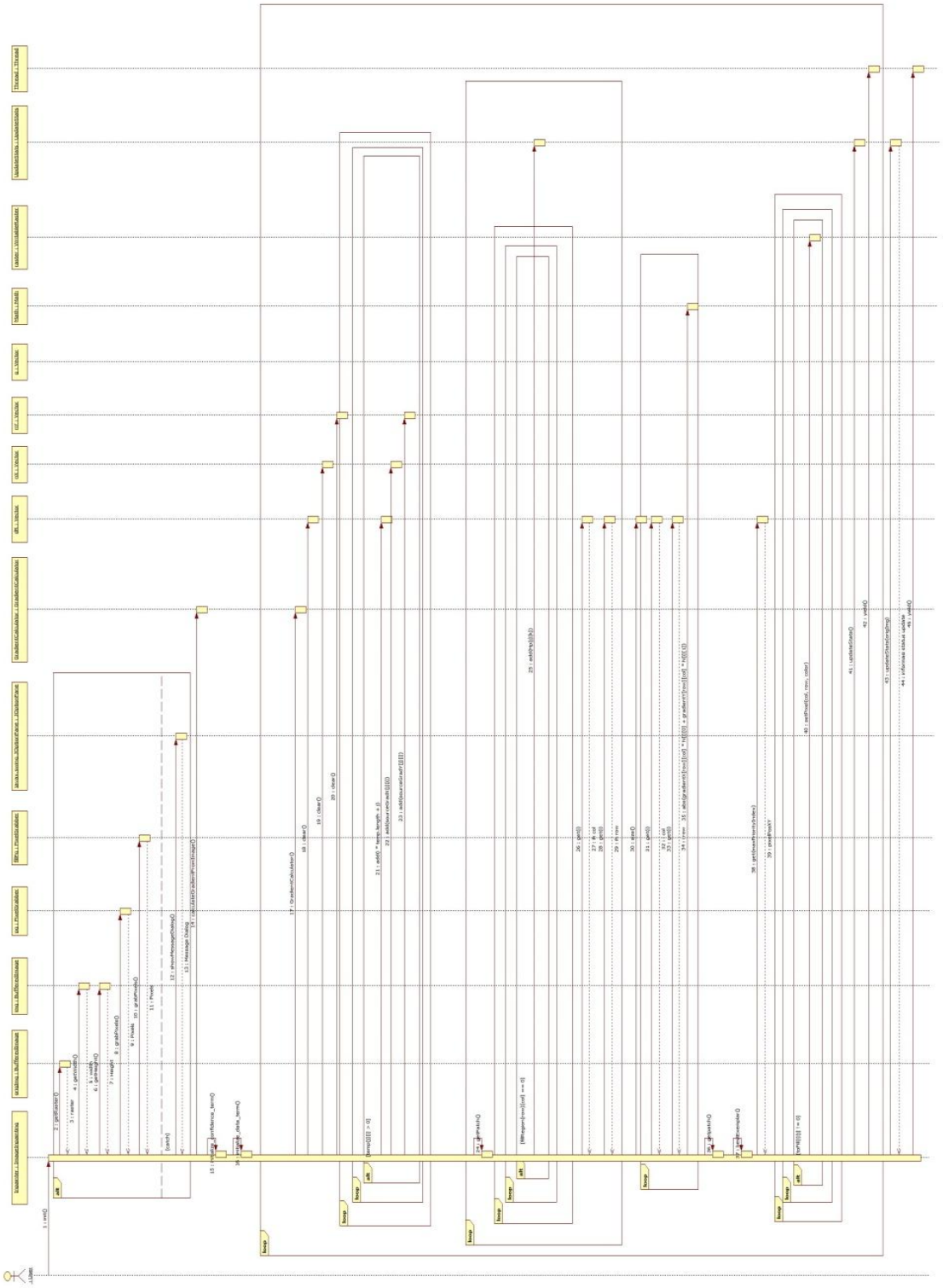
Gambar 3.14 *Sequence Diagram* Mengambil Gambar

2. *Sequence Diagram* Menandai Objek yang akan dihapus



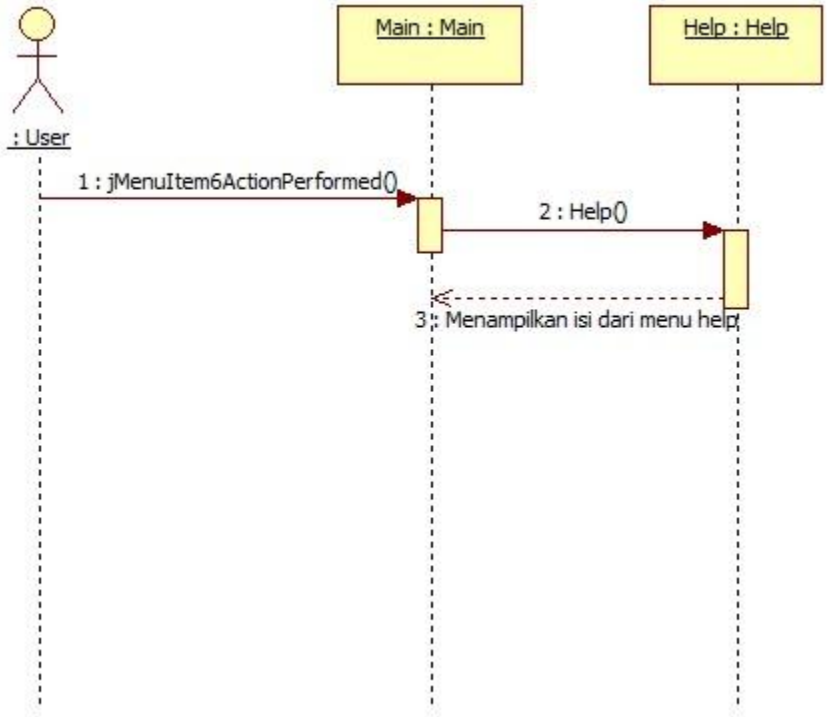
Gambar 3.15 *Sequence Diagram* Menandai Objek yang akan dihapus

3. *Sequence Diagram* Run *Inpainting*



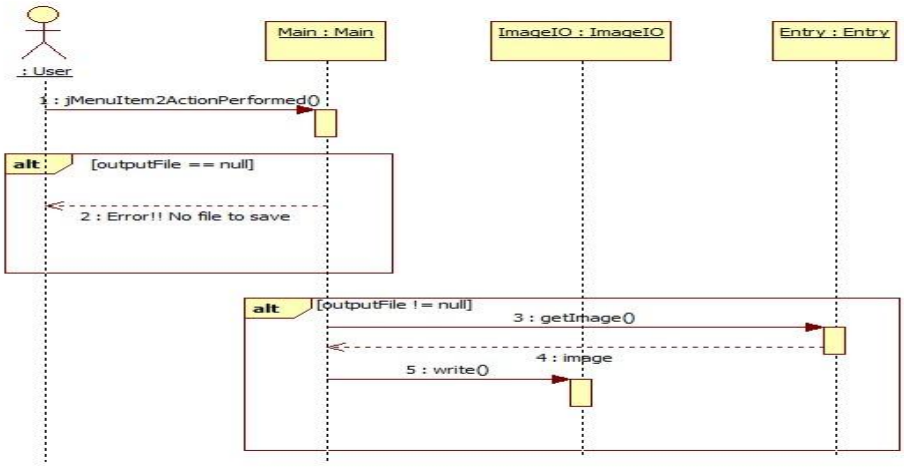
Gambar 3.16 *Sequence Diagram* Run *Inpainting*

4. *Sequence Diagram* Memilih Menu Bantuan



Gambar 3.17 *Sequence Diagram* Memilih Menu Bantuan

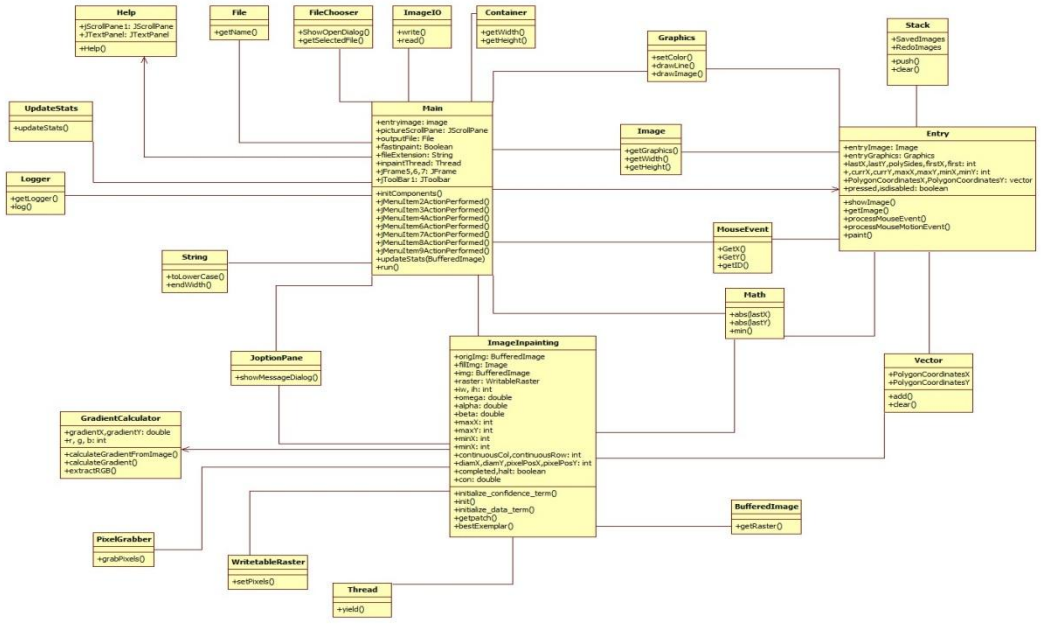
5. *Sequence Diagram* Memilih Menyimpan Gambar



Gambar 3.18 *Sequence Diagram* Menyimpan Gambar

3.3.4 *Class* Diagram

Class diagram digunakan untuk menggambarkan keadan suatu sistem dengan menjelaskan keterhubungan antara suatu class dengan class yang lain yang terdapat pada sistem. *Class* diagram yang digunakan pada pembangunan aplikasi ini yaitu :



Gambar 3.19 *Class* Diagram Aplikasi *Inpainting*

Tabel 3.9 Deskripsi *Class Diagram*

Class	Jenis Class	Deskripsi
Main	Interface	Merupakan *class* untuk menampilkan menu utama aplikasi.
Entry	Control	Merupakan *class* yang akan digunakan untuk menetukan daerah target.

ImageInpaint	Control	Merupakan *class* yang akan digunakan untuk memproses inapinting.
GradientCalculator	Control	Merupakan *class* yang berisi *method* untuk menentukan *gradient* dan mengekstrak RGB.
UpdateStats	Control	Merupakan *class* yang berisi *method update image* setelah dilakukan *inpainting*.
Help	Interface	Merupakan *class* untuk menampilkan menu bantuan.
File	Control	Mendefinisikan interface dan kelas untuk mesin virtual Java untuk mengakses file, atribut file, dan file sistem
FileChooser	Control	Merupakan *class* pemilih berkas untuk direktori *home user* yang hanya bisa dipilih berformat gambar.
ImageIO	Control	Merupakan *class* yang berisi metode kenyamanan statis untuk mencari *ImageReaders* dan *ImageWriters*, dan melakukan *encoding* dan *decoding* sederhana.
Container	Control	*Class Container* digunakan untuk memberikan dukungan default untuk aksesibilitas.
Graphics	Control	Merupakan *class* yang berisi semua

		gambar atau tulisan dilakukan dalam warna saat ini, menggunakan modus cat saat ini, dan dalam font saat ini.
Stack	Model	Merupakan *class* yang digunakan untuk menyimpan penyimpanan sementara proses.
String	Control	Merupakan *class* yang digunakan untuk mendapatkan ekstensi gambar yang sesuai dideklarasikan.
Logger	Control	Merupakan *logging* pesan akan diteruskan ke benda terdaftar *Handler*, yang dapat meneruskan pesan ke berbagai tujuan, termasuk konsol, file, OS log, dll
Image	Control	*Superclass* dari semua kelas yang mewakili gambar grafis
MouseEvent	Control	*Class* yang berisi *event* yang dilakukan *mouse*.
Math	Control	berisi metode untuk melakukan operasi numerik dasar seperti *elementary* eksponensial, logaritma, akar kuadrat, dan fungsi trigonometri.
JOptionPane	Control	Untuk memunculkan kotak dialog standar yang meminta pengguna untuk nilai atau menginformasikan sesuatu.
PixelGrabber	Control	Mengimplementasikan

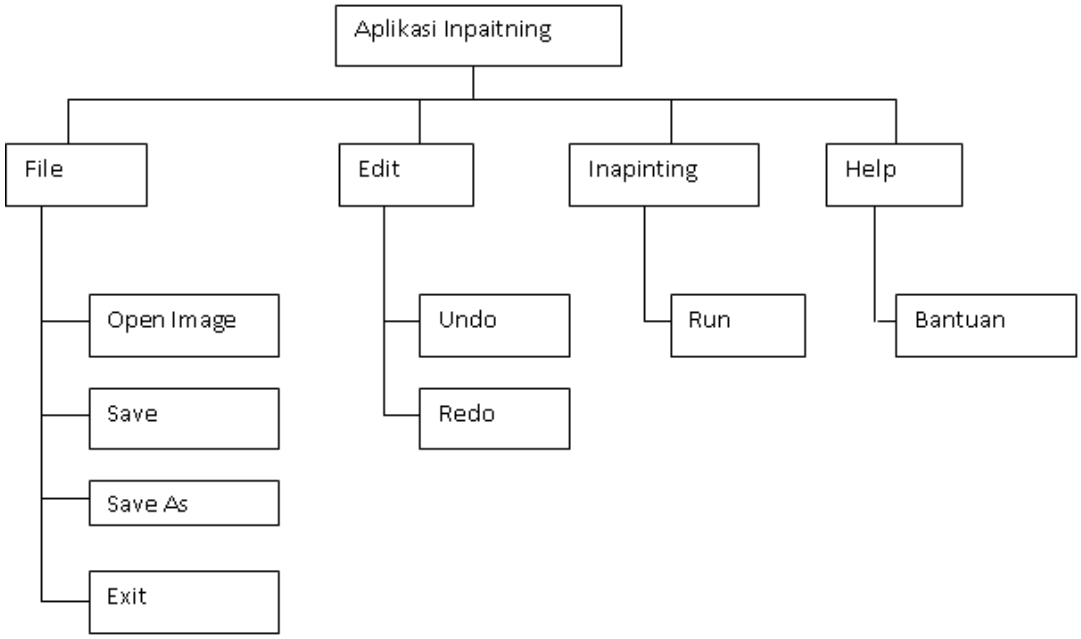
		ImageConsumer yang dapat dilampirkan ke gambar atau *ImageProducer* objek untuk mengambil subset dari piksel dalam citra.
WritetableRaster	Control	Untuk memberikan kemampuan menulis piksel.
Vector	Model	Mengimplementasikan *array growable* objek. Seperti array, berisi komponen yang dapat diakses menggunakan indeks integer.
Thread	Control	*Thread* eksekusi dalam sebuah program. *Java Virtual Machine* memungkinkan aplikasi untuk memiliki beberapa *thread* eksekusi berjalan secara bersamaan.
BufferedImage	Control	*Subclass* menggambarkan sebuah gambar dengan *buffer* diakses data citra. Sebuah *BufferedImage* terdiri dari *ColorModel* dan Raster data citra.

3.4 Perancangan Sistem

Perancangan sistem merupakan suatu proses yang menggambarkan bagaimana suatu sistem dibangun untuk memenuhi kebutuhan pada fase analisis. Tahap perancangan berkaitan dengan kegiatan implementasi seperti perancangan struktur menu, perancangan antarmuka, dan jaringan semantik.

3.4.1 Perancangan Struktur Menu

Perancangan menu dilakukan untuk mempermudah interaksi antara sistem dengan pengguna. Perancangan struktur menu aplikasi ini dapat dilihat sebagai berikut :



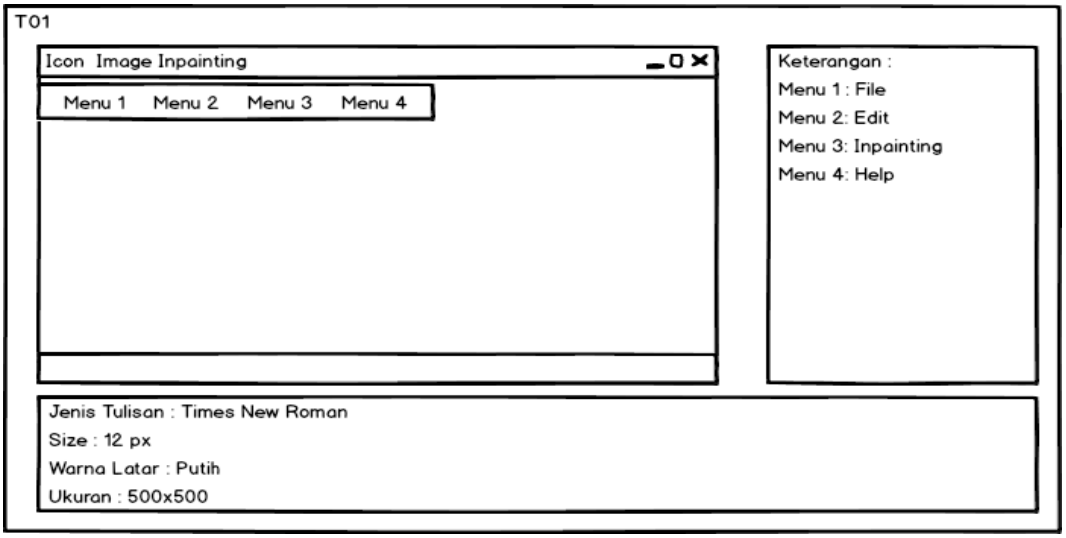
Gambar 3.20 Struktur Menu Aplikasi *Inpainting*

3.4.2 Perancangan Antarmuka

Perancangan antarmuka adalah tahapan pembuatan rancangan antarmuka untuk digunakan pada pembangunan aplikasi ini yang dibagi menjadi empat menu yaitu file, edit, *inpainting*, dan help.

1. Perancangan Antarmuka Utama

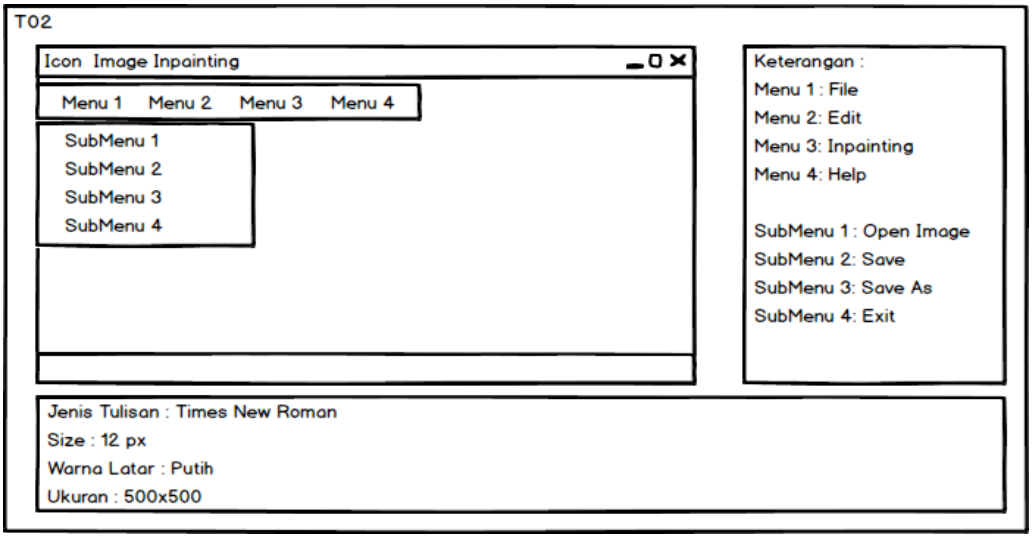
Perancangan antarmuka menu yang tampil waktu pertama program berjalan. Dimana ada empat menu yaitu file, edit, inpainting, dan help.



Gambar 3.21 Perancangan Antarmuka Menu Uama

2. Perancangan Antarmuka Menu File

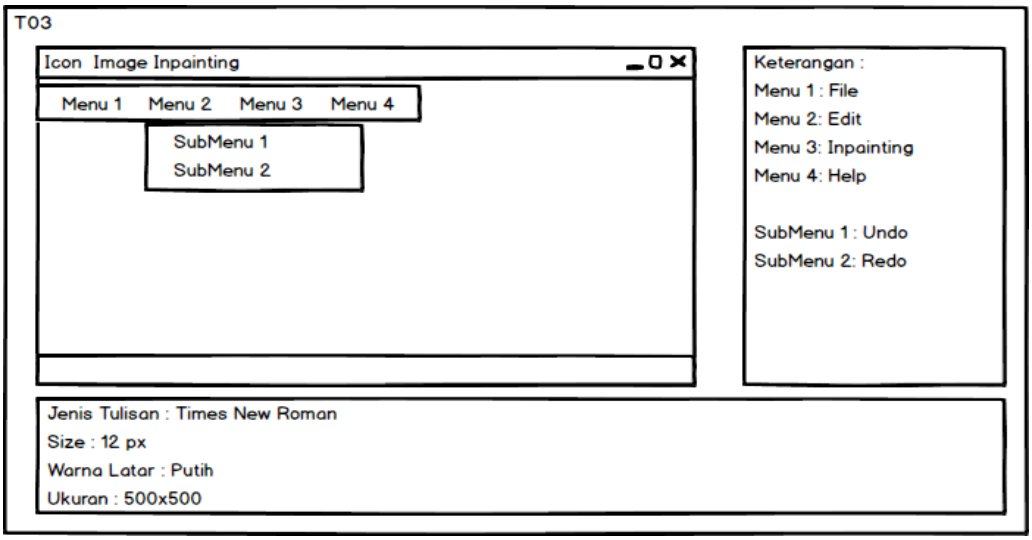
Perancangan antarmuka menu yang tampil di menu file. Di menu file ini terdapat 4 submenu yaitu : Open Image, Save, Save As, dan Exit.



Gambar 3.22 Perancangan Antarmuka Menu File

3. Perancangan Antarmuka Menu Edit

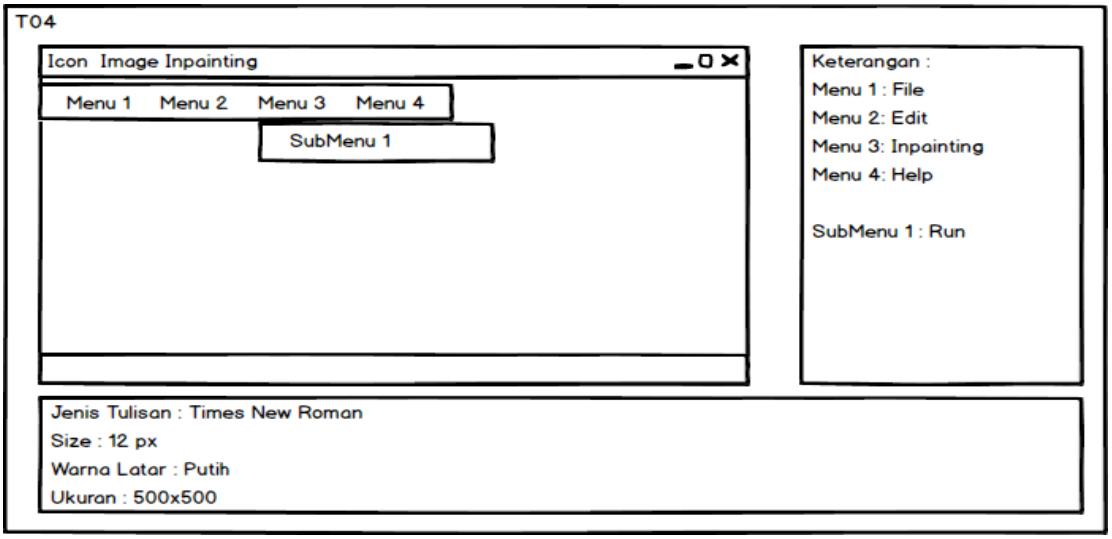
Perancangan antarmuka menu yang tampil di menu Edit. Di menu file ini terdapat 2 submenu yaitu : *Undo* dan *Redo*.



Gambar 3.23 Perancangan Antarmuka Menu Edit

4. Perancangan Antarmuka Menu Inpaint

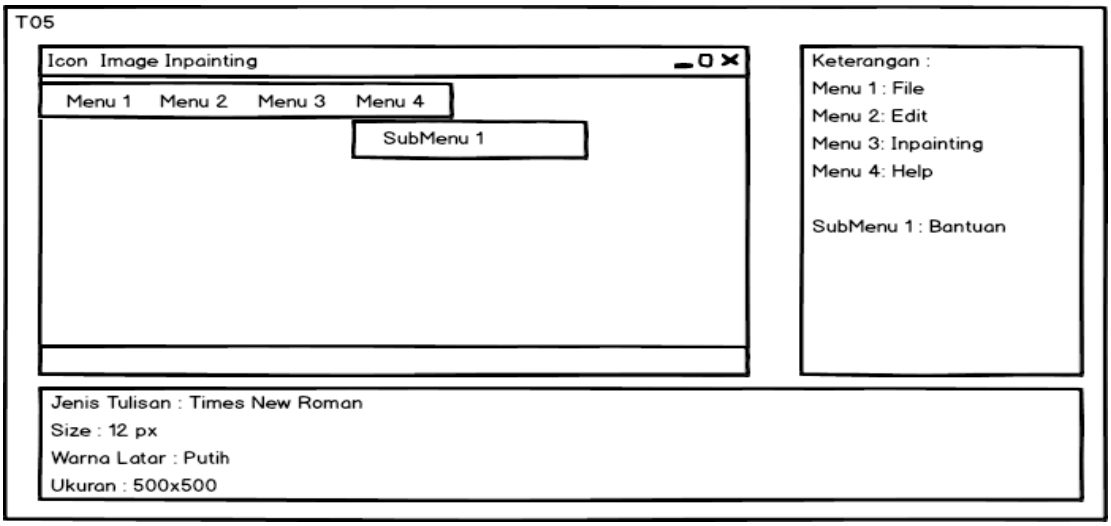
Perancangan antarmuka menu yang tampil di menu Inpaint. Di menu file ini terdapat 1 submenu yaitu : Run.



Gambar 3.24 Perancangan Antarmuka Menu Inpaint

5. Perancangan Antarmuka Menu Help

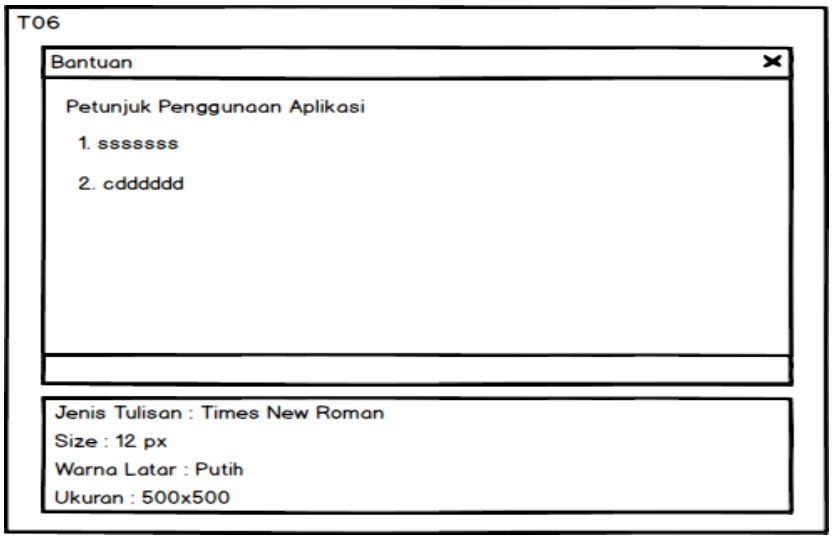
Perancangan antarmuka menu yang tampil di menu Help. Di menu file ini terdapat 1 submenu yaitu : Bantuan.



Gambar 3.25 Perancangan Antarmuka Menu Help

6. Perancangan Antarmuka Submenu Bantuan

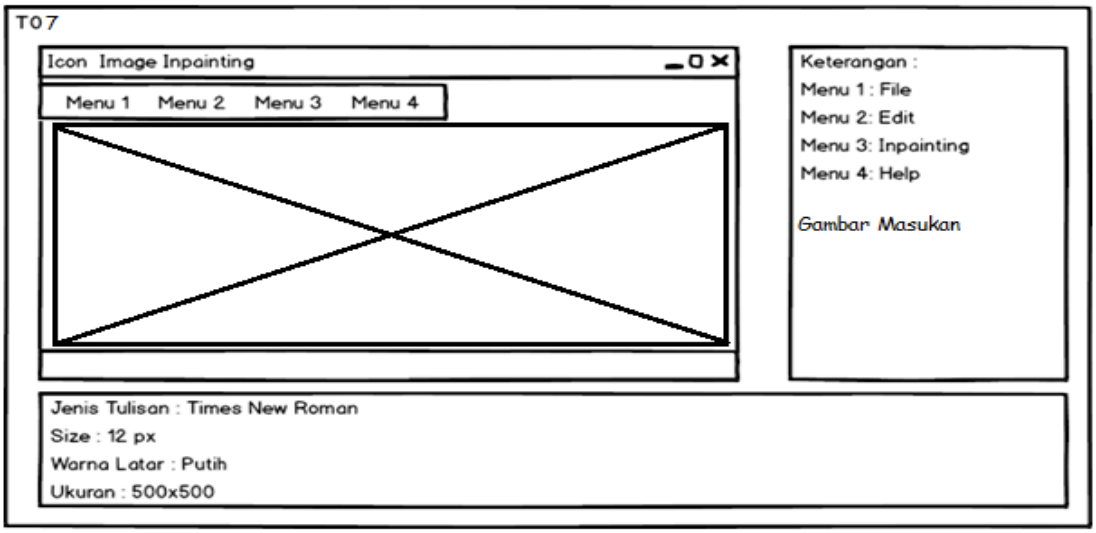
Perancangan antarmuka menu yang tampil setelah *user* memilih submenu Bantuan di menu Help yang berisikan petunjuk penggunaan aplikasi.



Gambar 3.26 Perancangan Antarmuka Bantuan

7. Perancangan Antarmuka Citra Masukan

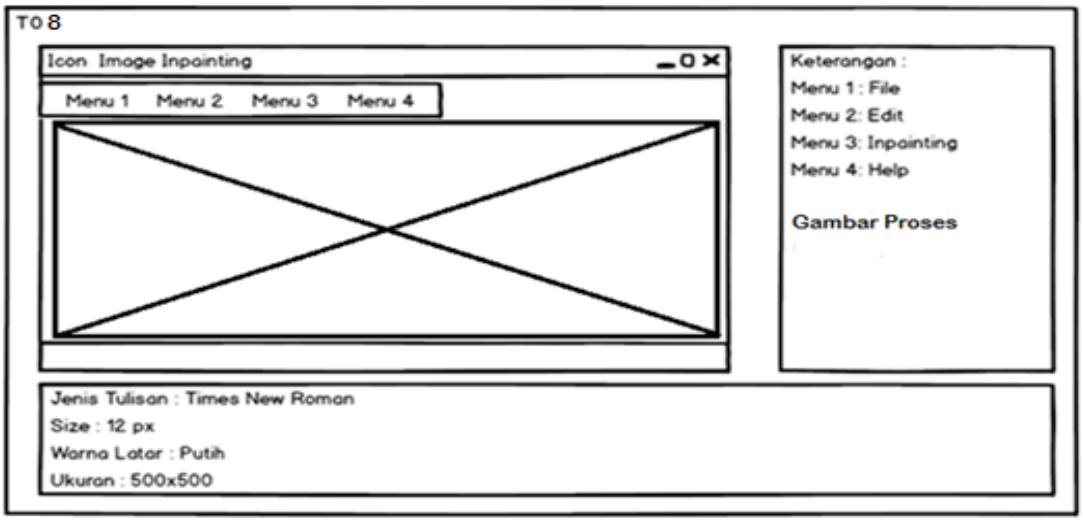
Perancangan antarmuka yang tampil setelah *user* memilih citra dari galeri.



Gambar 3.27 Perancangan Antarmuka Citra Masukan

8. Perancangan Antarmuka Proses Restorasi

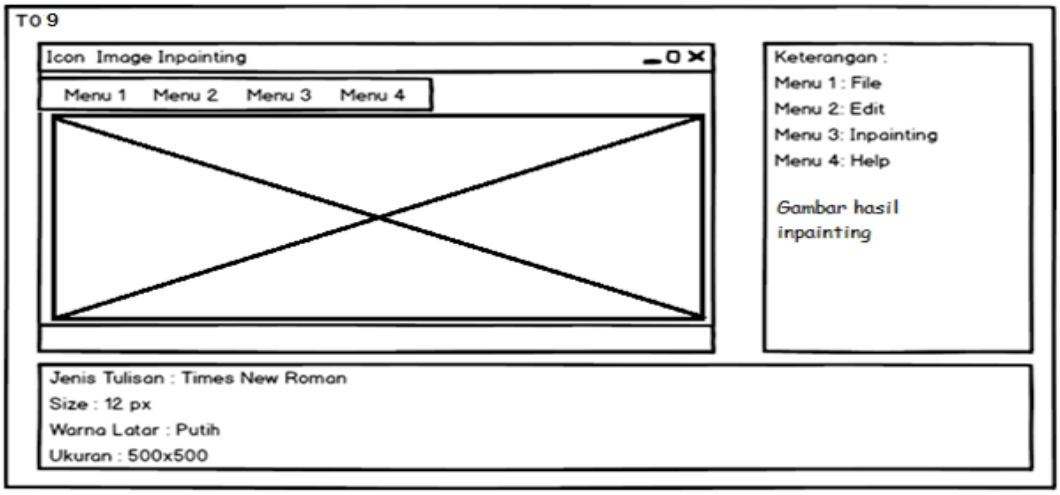
Perancangan antarmuka yang tampil setelah *inpainting* selesai atau citra yang sudah direstorasi.



Gambar 3.28 Perancangan Antarmuka Proses Restorasi

9. Perancangan Antarmuka Citra Hasil Inpainting

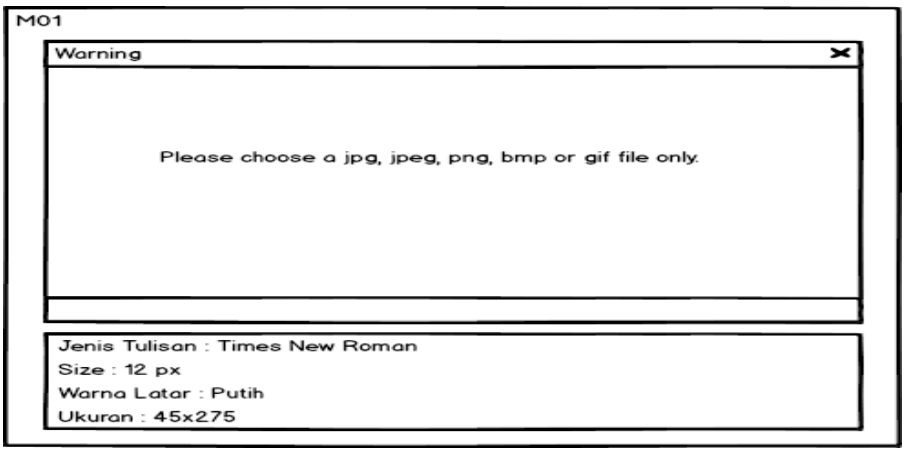
Perancangan antarmuka yang tampil setelah *inpainting* selesai atau citra yang sudah direstorasi.



Gambar 3.29 Perancangan Antarmuka Citra Hasil

10. Perancangan pesan kesalahan jika memasukan file yang bukan merupakan gambar

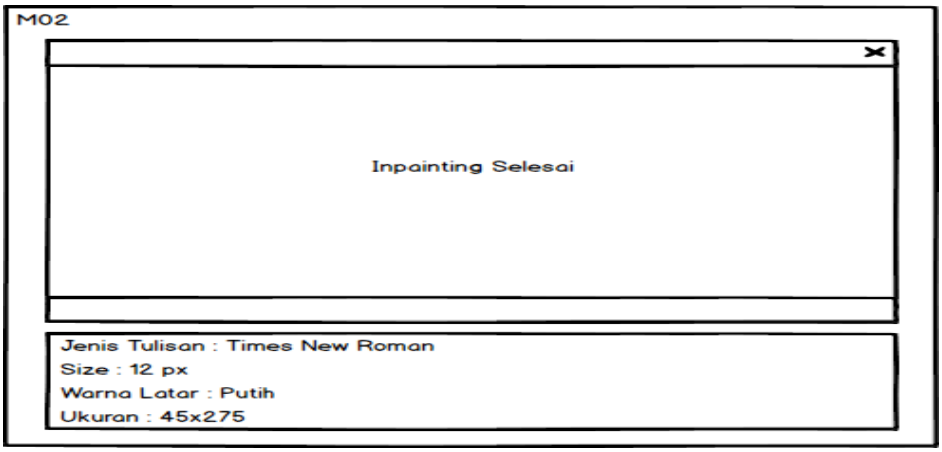
Perancangan antarmuka pesan error jika *user* setelah memilih submenu open image di menu di file dengan memasukan file yang bukan gambar.



Gambar 3.30 Perancangan Antarmuka Pesan jika *user* memasukan bukan file gambar

11. Perancangan pesan jika proses sudah selesai

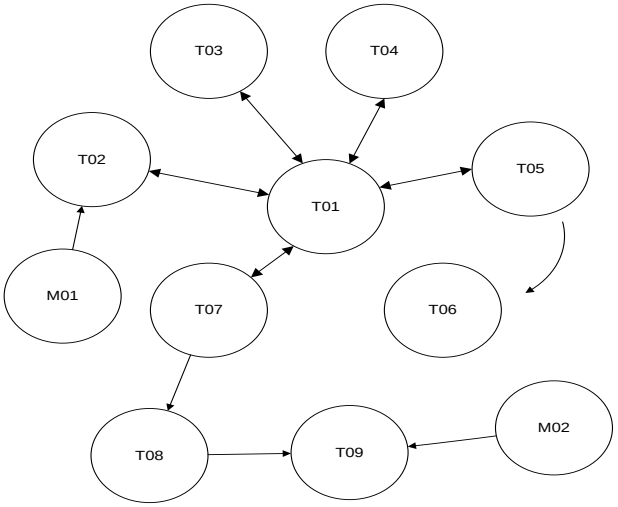
Perancangan antarmuka pesan setelah proses *inpainting* selesai.



Gambar 3.31 Perancangan Antarmuka Pesan setelah proses berlangsung

3.4.3 Jaringan Semantik

Jaringan semantik adalah diagram yang menggambarkan aliran-aliran menu dan pesan dalam sebuah program. Jaringan semantik dari aplikasi ini yang dibangun yaitu



Gambar 3.32 Jaringan semantik Aplikasi *Inpainting*