# BAB III

# PERANCANGAN SISTEM

## 3.1 Analisis Sistem

Sistem yang akan dibuat adalah "Pendeteksi Sabotase Gambar Digital sebagai Digital Image Forensik". Sistem ini merupakan aplikasi berbasis web yang berjalan pada sebuah *web browser* dengan komunikasi menggunakan protokol HTTP. Aplikasi ini mampu mendeteksi adanya objek dalam sebuah gambar digital yang merupakan duplikat dari objek lain yang ada di dalam gambar digital tersebut. proses pendeteksian tersebut dilakukan dengan metode *"Robust Algorithm"*.

Terdapat kendala ketika seseorang ingin mendapatkan informasi dari sebuah gambar digital, karena bisa saja gambar digital tersebut telah disabotase dalam hal *content,* misalnya kita terkadang mendapatkan kesulitan dalam memastikan keaslian jumlah objek dalam sebuah gambar digital buah – buahan, ataupun adanya sesesorang yang ditutupi oleh objek pohon.

Dari permasalahan yang muncul tersebut terdapat solusi yaitu dengan membuat sebuah aplikasi berbasis web yang mampu mendeteksi adanya sabotase gambar digital yang berjalan pada sebuah *web browser* komputer.

Aplikasi tersebut dibangun dengan mengacu pada algoritma *ad-hoc* yang telah termasuk pada algoritma *robust*. Algoritma ini membutuhkan inisialisasi segmen awal yang dijadikan sebagai nilai ukuran blok piksel NxN yaitu 15 x 15 piksel, namun sebelum itu gambar digital yang akan diproses terlebih dahulu melalui proses *blurring* dengan level yang dapat diset secara *custom*. Selain blur level, *user* juga bisa melakukan set terhadap level *threshold*. Ada juga beberapa ukuran daerah identik dari blok yang akan dilakukan proses *threshold* yang bisa diset oleh *user*.

## 3.2 Analisis Kebutuhan Non-Fungsional

Dalam sebuah sistem terdapat beberapa komponen yang bersifat non-fungsional. Komponen ini meliputi *hardware, software, user* atau pengguna.

### *3.2.1 Hardware*

Program yang dibuat akan dirancang berjalan di dalam sebuah *web browser* komputer. Adapun kebutuhan *hardware* dalam membangun serta menjalankan aplikasi ini adalah sebagai berikut :

1. Prosesor : pentium D 2.66 GHz.
2. RAM : 2 GB DDR2.
3. Hardisk : SATA 2.
4. Monitor : 17 inch. CRT.

Pada dasarnya program ini tidak membutuhkan spesifikasi sistem komputer secara khusus, artinya dapat berjalan pada spesifikasi sistem komputer apa saja, namun akan terdapat perbedaan dalam menyelesaikan proses deteksi *copy-move* gambar digital jika program ini dijalankan pada spesifikasi komputer yang memiliki prosesor dengan kecepatan tinggi, karena aplikasi ini banyak menggunakan *resource* prosesor komputer.

### *3.2.2 Software*

Jika ditinjau dari segi kebutuhan *software* apa saja yang dibutuhkan oleh program ini meliputi, *python* versi 2.6 yang harus terinstal dalam sebuah komputer yang ingin melakukan deteksi *copy-move* gambar digital, serta sebuah library *PIL (Python Image Library) 1.1.7.* Adapaun perangkat *server* yang digunakan adalah XAMPP 3.0.12. Berbeda dengan kebutuhan *hardware* yang bersifat fleksibel, dalam hal kebutuhan non-fungsional *software* ini bersifat mutlak yang tidak bisa ditolerir, artinya semua persyaratan yang ada pada kebutuhan *software* harus terpenuhi.

*Python 2.6* berfungsi sebagai interpreter dari *source code* yang telah dibuat. Interpreter tersebut akan melakukan tugasnya tergantung di sistem mana python

tersebut di instal. Sistem dalam hal ini bisa dibilang adalah sebuah sistem operasi. Jika sistem tidak memiliki *python* maka akibatnya akan fatal seperti aplikasi tidak berjalan sama sekali.

Pada *core* program terdapat proses *"import"* sebuah *library*. *Library* merupakan sekumpulan fungsi yang telah siap digunakan. Maka dari itu pada sistem python mutlak adanya *library*. Sama halnya dengan *python* jika *library* ini tidak ada pada sistem *python*, maka aplikasi tidak akan berjalan.

### *3.2.3 User* atau Pengguna

Aplikasi ini ditujukan kepada orang – orang yang ingin melakukan investigasi terhadap keaslian dari sebuah gambar digital, terkhusus dalam pemeriksaan ada tidaknya objek yang telah diduplikat. Pengunaan aplikasi ini jika ditinjau dari segi umur maupun profesi tidak terbatas, artinya aplikasi ini bebas digunakan oleh siapa saja yang mengerti tentang pengolahan gambar karena adanya parameter yang harus disetoleh *user*. Namun secara khusus aplikasi ini ditujukan bagi pihak berwajib yang ingin melakukan pembuktian terhadap adanya perkiraan sabotase gambar digital.

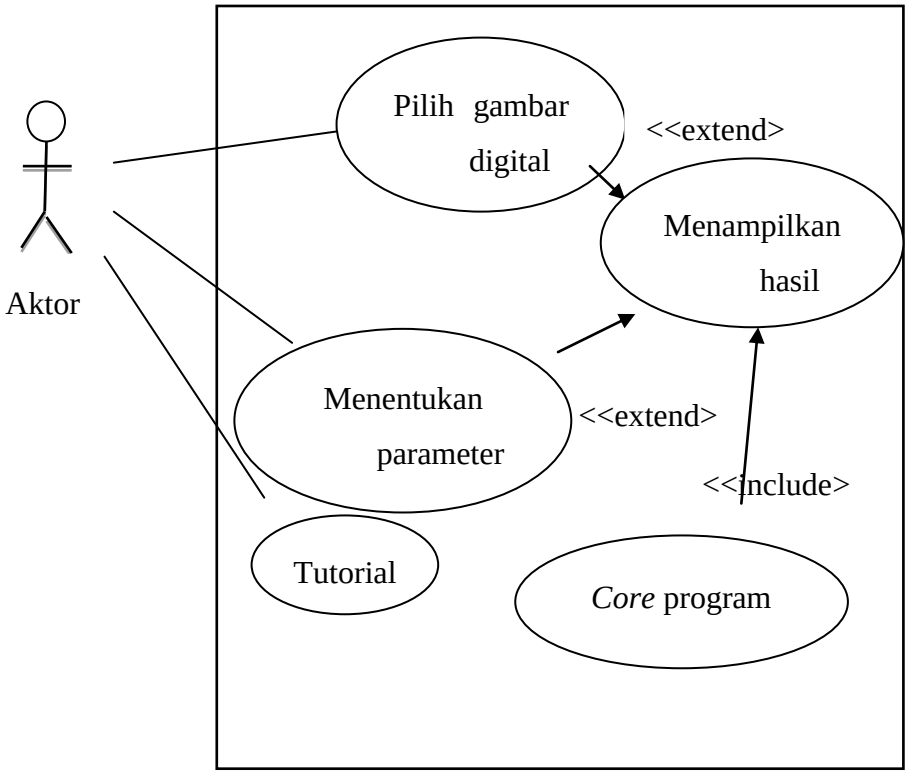
## 3.3 Analisis Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan yang terdiri dari berbagai diagram yang ada pada pemrograman UML. Pada aplikasi yang ingin dibuat setidaknya membutuhkan empat diagram yang terdiri dari *use case diagram, activity diagram, class diagram*serta *sequence diagram*.

### 3.3.1 *Use Case Diagram*

Sistem yang dirancang memiliki tiga buah *use case*, yang terdiri dari penentuan lokasi gambar digital, penentuan parameter serta menampilkan hasil proses. Namun aktor dalam hal ini hanya dapat menentukan lokasi gambar digital serta menentukana parameter setelah itu aktor mengirimkan ke *server* untuk segera diproses. Dalam hal ini aktor bisa siapa saja, karena yang dibutuhkan oleh aktor hanya sedikit pengetahuan tentang pengolahan

gambar digital, seperti *threshold*, *blur* dan sebagainya. Gambar 3.1 di bawah ini menjelaskan sistem *use case* yang akan dibuat :



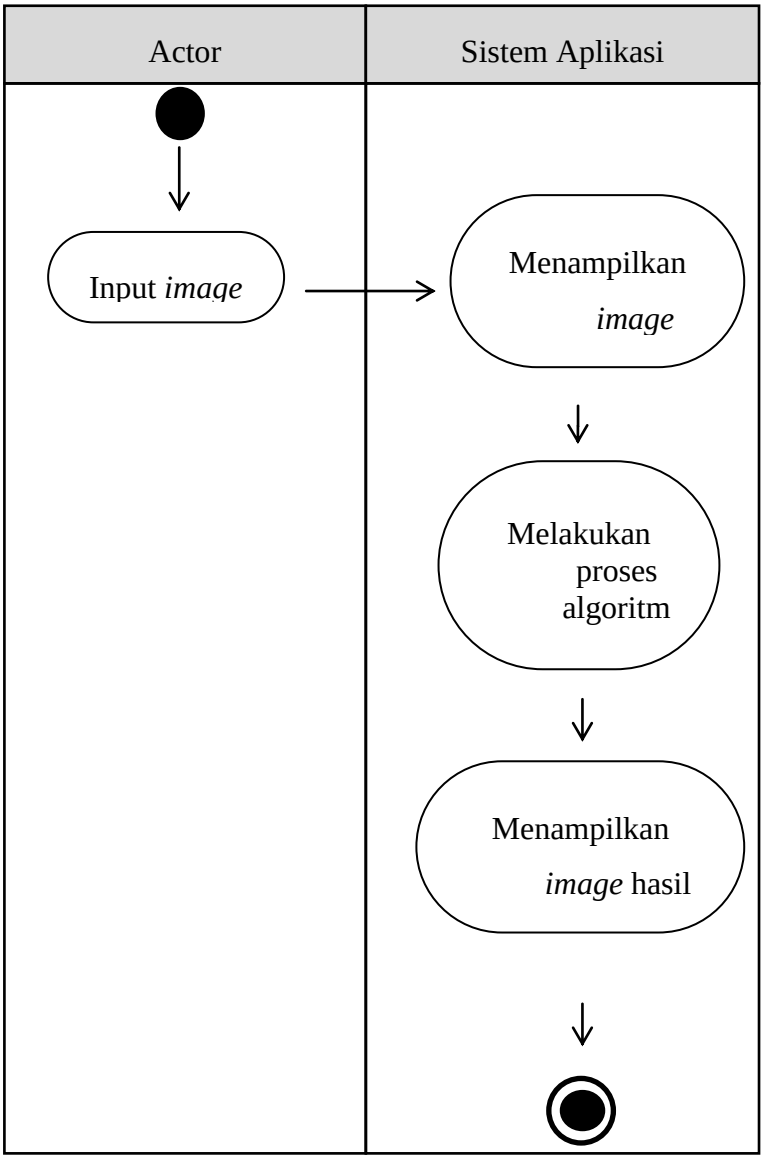

Gambar 3.1. *Use case diagram*

Nilai parameter tersebut penting diperhatikan oleh *user* yang akan menggunakan program tersebut, karena parameter ini akan menentukan cara kerja dari program. Parameter – parameter ini dibuat untuk menentukan nilai dari level *threshoold* serta besar blok. Namun parameter tersebut telah memiliki nilai *default*-nya, jadi bisa saja aktor tidak menentukan nilai parameter, tapi hanya menentukan lokasi gambar digital saja lalu langsung memprosesnya.

### 3.3.2 *Activity Diagram*

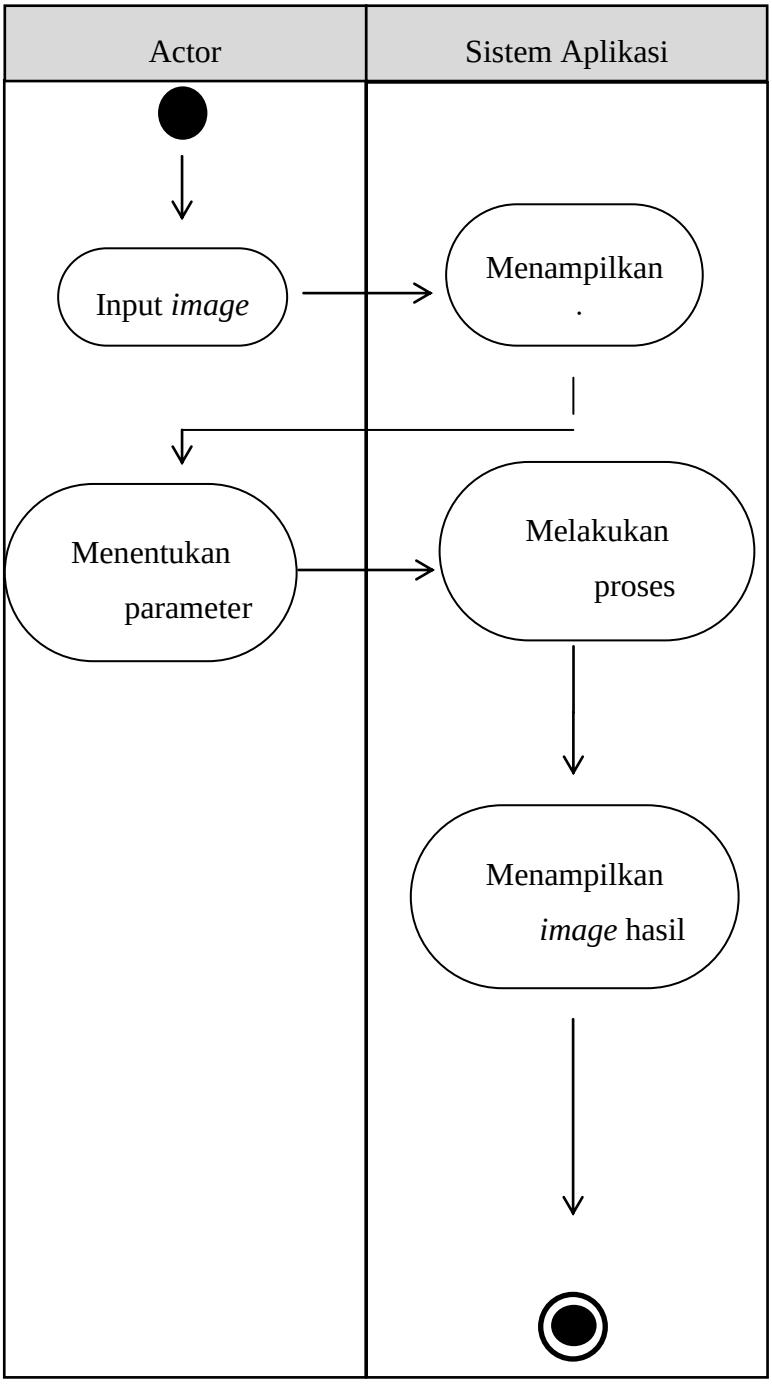
*Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing – masing alir berawal, *decision*

yang mungkin terjadi, dan bagaimana mereka berakhir. Berikut adalah *activity diagram* dari salah satu proses yang mungkin terjadi :




Gambar 3.2. *Activity diagram* pilih gambar

*Activity diagram* yang pertama ini melakukan proses tanpa pengaturan parameter terlebih dahulu. Pada gambar 3.2 terlihat bahwa *user* hanya melakukan input *image*, setelah itu *image* yang telah dipilih tersebut langsung diproses untuk dideteksi adanya sabotase gambar digital.

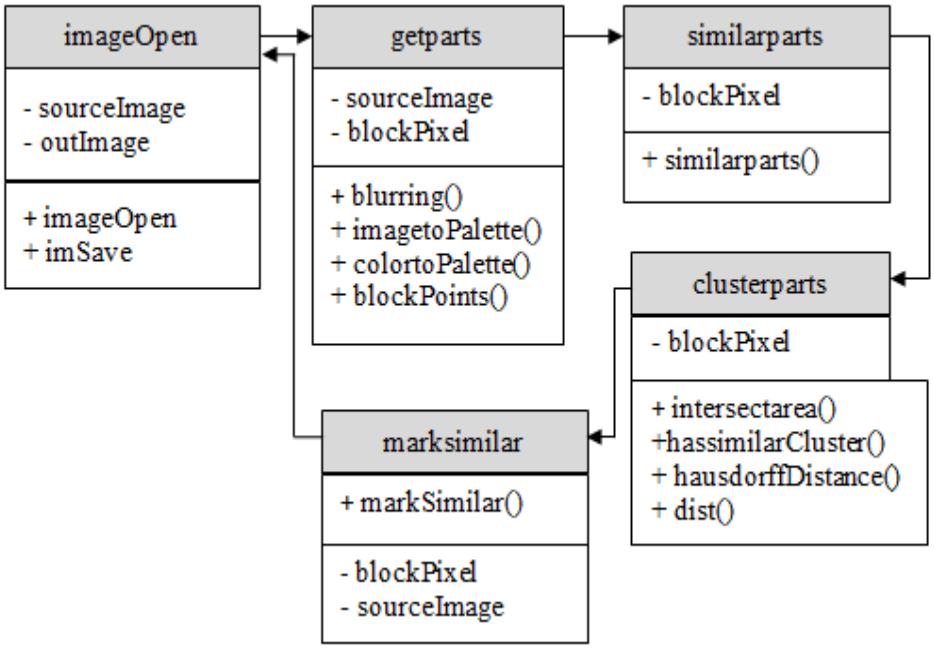



Gambar 3.3. *Activity diagram* menentukan parameter

Berbeda pada *activity diagram* gambar 3.2, yang terjadi pada *activity diagram* gambar 3.3, user akan merubah nilai *default* parameter. Nilai ini diperuntukkan sebagai inisialisasi untuk aplikasi.

### 3.3.3 *Class Diagram*

Diagram *Class* digunakan untuk menggambarkan secara abstrak struktur dari aplikasi yang akan dibangun, *class – class* yang terlibat, serta hubungan antar *class* untuk saling berkomunikasi satu sama lain. Struktur *class* hanya ada pada *core* program. Berikut *class* diagram dari aplikasi yang akan dirancang :

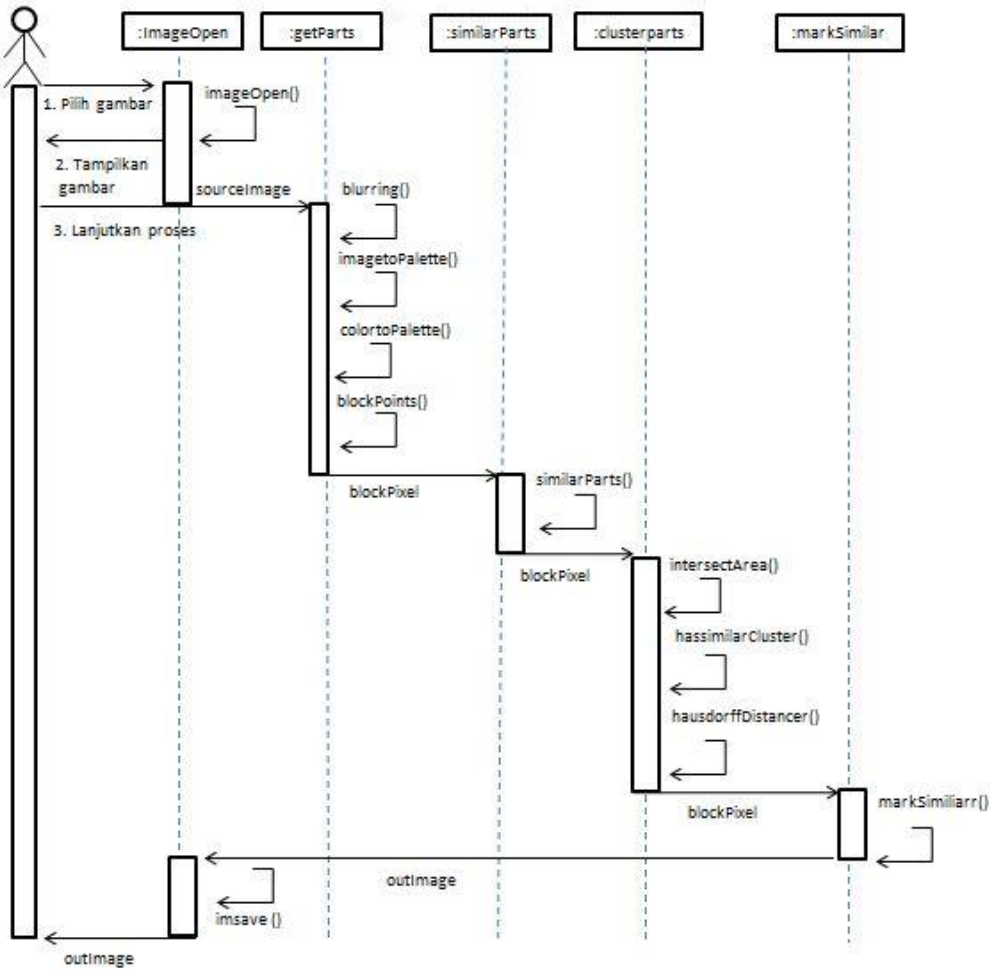



Gambar 3.4. *Class diagram*

Aplikasi yang dibangun setidaknya memiliki empat *class*, yang setiap *class*-nya memiliki kemungkinan memanggil *sub-class*. *Class* utama bernama *"imageOpen"* yang berperan mengirimkan gambar digital untuk diproses oleh *class – class* lainnya, setelah seluruh *class* melakukan tugasnya masing – masing maka *class imageOpen* akan menerima gambar digital untuk di *save.*

### 3.3.4 *Sequence Diagram*

*Sequence diagram* akan mempermudah kita dalam merancang aplikasi yang akan kita buat. Diagram ini dibagi ke dalam 2 sudut pandang, secara horizontal dan vertikal. Berikut gambar 3.5 adalah *sequnce diagram* dari aplikasi yang akan dibuat :

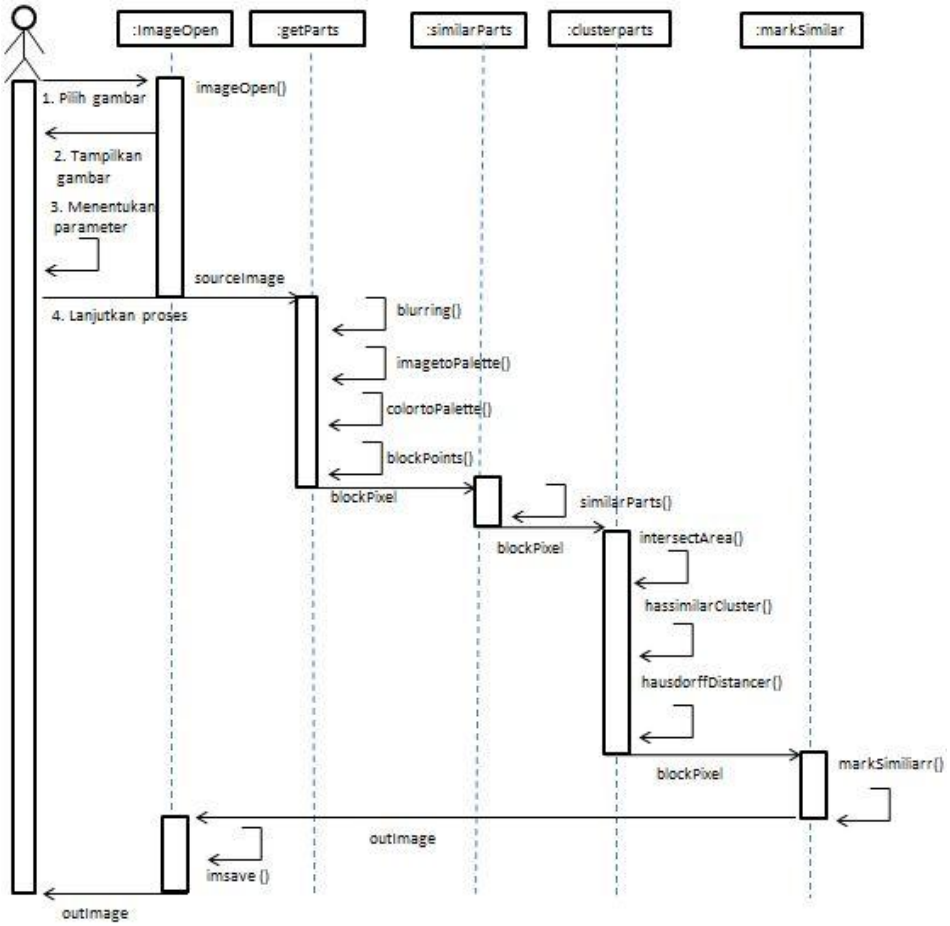



Gambar 3.5. *Sequence diagram* parameter *default*

Pada *sequence diagram* parameter default yang terdapat pada gambar 3.5 merupakan *diagram sequence* yang pertama. Pada *sequence diagram* ini *user* tidak perlu merubah setting parameter, karena *core* program telah memiliki nilai default.

Dimensi horizontal yang mewakili objek yang terlibat di dalam sebuah sistem, diantaranya *actor* yang berperan memilih gambar digital yang akan

dilakukan proses forensik, lalu dimensi vertikal meliputi aplikasi *"Digital Image Forensics"*, untuk memproses gambar digital.




Gambar 3.6. *Sequence diagram* menentukan parameter

Berbeda dengan sequence diagram pada parameter *default*, pada *seqeunce* diagram menetukan parameter ini, *user* bisa merubah setting parameter untuk mendapatkan hasil perkiraan yang maksimal dari aplikasi ini.

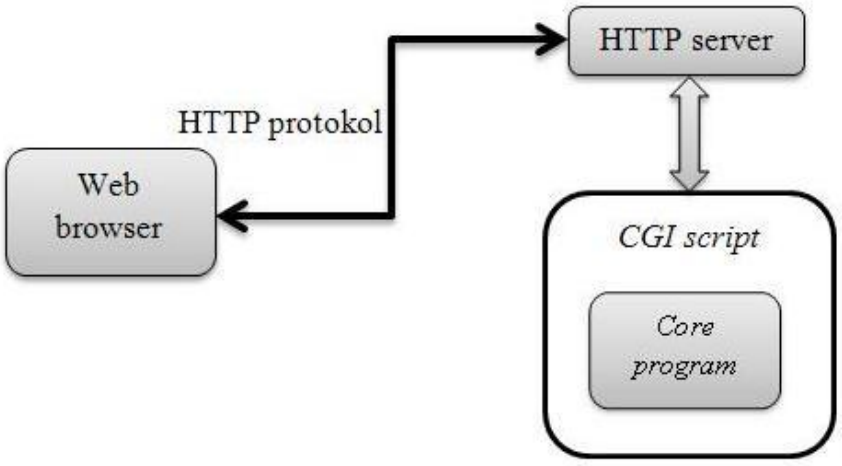
## 3.4 Perancangan Sistem

Perancangan sistem ditujukan sebagai acuan dalam membangun sistem aplikasi yang akan dibuat. Ada beberapa perancangan yang harus dipahami terlebih dahulu yaitu perancangan aplikasi, perancangan antarmuka serta prosedural.

### 3.4.1 Perancangan Aplikasi

Perangkat lunak yang ingin dirancang telah memiliki GUI *(Graphical User Interface)* sebagai antarmuka antara *user* dengan fungsi – fungsi algoritma yang ada di dalam program. Fungsi – fungsi algoritma tersebut disusun dengan menggunakan metode pemrograman berorientasi objek *(object oriented programming)* yang mengacu pada UML *(Unified Modelling Language).* Berbeda dengan pemrograman terstruktur, UML pada OOP telah memiliki sejumlah *class* yang dapat digunakan untuk fungsinya masing – masing.

Secara umum konsep sistem yang akan dibuat dapat dilihat pada gambar diagram blok di bawah ini :



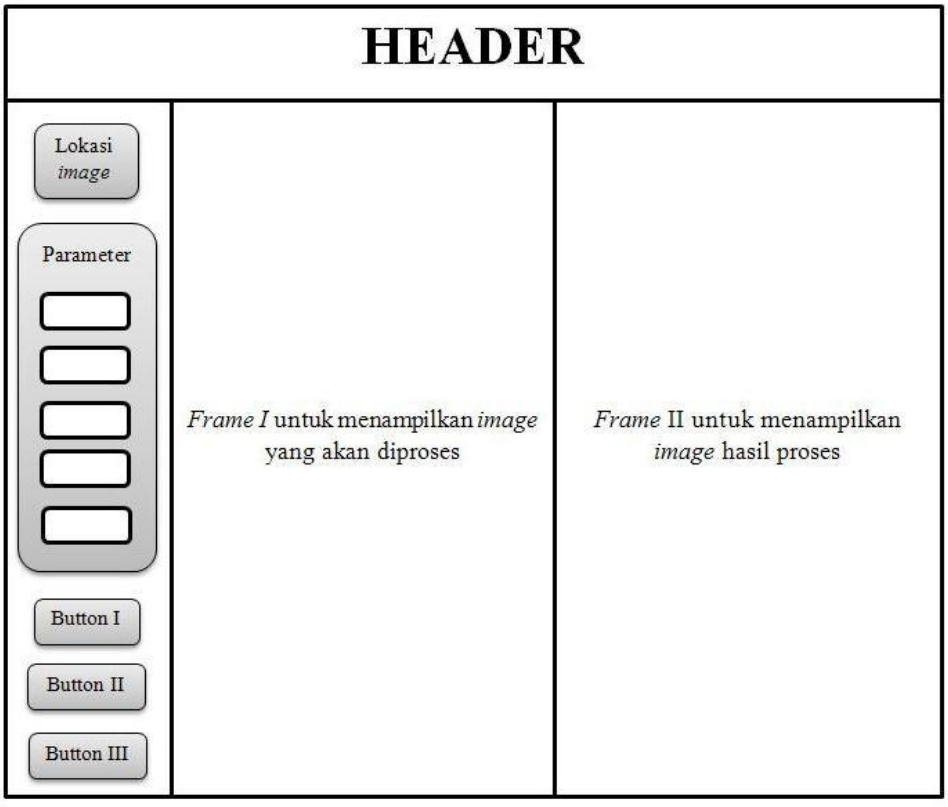

Gambar 3.7. Diagram Blok Sistem

Diagram blok pada gambar 3.7 menjelaskan alur data pada program yang akan dibuat. Alur data tersebut bermulai dari *web browser* yang diinisialisasi oleh *user*, yaitu berupa *directory local* sebuah gambar digital serta parameter – parameternya. Setelah itu *directory* serta parameter dikirim melalui HTTP protokol dan di terima oleh *server*. Di dalam *server*, *directory* serta parameter tersebut “ditampung” oleh *CGI script* untuk selanjutnya diberikan kepada *core program* untuk dapat diproses. Setelah proses mendeteksi selesai, *CGI script* akan

memberikan *directory* tempat menyimpan gambar digital yang telah diproses tadi untuk ditampilkan dihalaman *website* melalui HTTP protokol.

### 3.4.2 Perancangan Antarmuka

Perangkat lunak yang ingin dirancang telah memiliki GUI *(Graphical User Interface)* sebagai antarmuka antara *user* dengan fungsi – fungsi algoritma yang ada di dalam program. Fungsi – fungsi algoritma tersebut disusun dengan menggunakan metode pemrograman berorientasi objek *(object oriented programming)* yang mengacu pada UML *(Unified Modelling Language)*. Berikut adalah gambaran antarmuka yang dibuat :




Gambar 3.8. Antarmuka Aplikasi *web-based*

Adapun penjelasan dari setiap bagian antarmuka aplikasi adalah sebagai berikut :

1. HEADER : merupakan nama dari aplikasi.

2. Lokasi *image* : tombol untuk memilih gambar digital.
3. Parameter : angka yang secara *optional* bisa di-*set* oleh *user*.
4. Button I : tombol "Proses" untuk mengirimkan lokasi serta parameter gambar digital.
5. Button II : tombol "Reset" untuk mengembalikan nilai parameter ke semula *(default)*.
6. Button III : tombol tutorial sebagai panduan *user* baru.
7. *Frame*I : digunakan untuk menampilkan gambar digital yang akan diproses.
8. *Frame* II : digunakan untuk menampilkan gambar digital yang telah diproses.

Antarmuka program didesain untuk memudahkan user dalam menggunakan aplikasi *software* yang kita buat. Antarmuka program juga berguna untuk menerima informasi yang ingin didapat dari *user* untuk digunakan sebagai parameter dalam program yang dibuat.

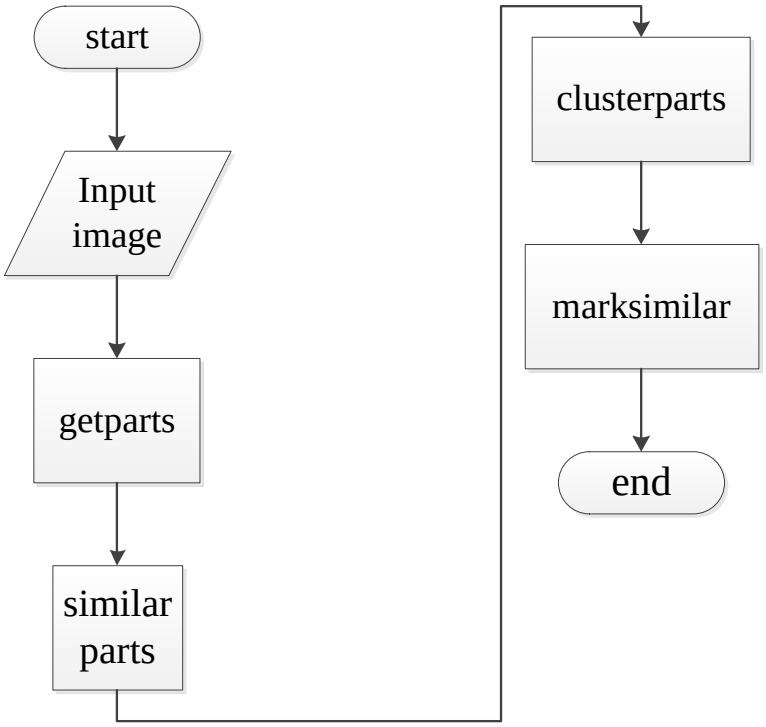
### 3.4.3 Prosedural

Pembuatan aplikasi digital *image* forensik ini mengacu pada sebuah metode algoritma yang disebut *"Robust Algorithm"*. Metode algoritma ini memiliki metode meng-ekstrak sebuah gambar digital menjadi blok – blok piksel yang nantinya akan dicari blok piksel yang identik dari sejumlah blok – blok matriks piksel yang ada.

Berikut langkah – langkah dari metode *"Robust Algorithm"*:

1. *Blur image.*
2. *Convert image* ke dalam palet.
3. Susun *image* ke dalam blok – blok piksel kecil NxN.
4. Disusun berdasarkan nilai piksel.
5. Urutkan blok – blok yang berdekatan serta memiliki perbedaan warna yang mutlak.
6. Ekstrak kelompok – kelompok yang memiliki kesamaan nilai piksel.
7. Tandai kelompok blok yang sama pada *image* yang telah ditemukan.

Jika digambarkan *folwchart* dari algoritma *robust* yang nantinya akan dijadikan acuan dalam pembuatan aplikasi adalah sebagai berikut :




Gambar 3.9. *flowchart* Algoritma *Robust*

#### *3.4.3.1 Getparts*

Pada proses *getparts* berarti gambar digital akan dipecah menjadi blok – blok piksel yaitu sebesar 15x15 piksel. Namun sebelum proses tersebut dilakukan terlebih dahulu melalui proses *blurring*. Proses ini dilakukan untuk menghilangkan detil pada gambar digital. Proses *blurring* ini dilakukan dengan menggunakan metode *"Gaussian"*, dengan rumus persamaan sebagai berikut :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{..... (III.1)}$$

Dimana x adalah jarak dari titik asal pada sumbu horizontal, y adalah jarak dari titik asal pada sumbu vertikal, dan σ adalah standar deviasi dari distribusi *gaussian*.

Setelah melalui proses *blurring*, gambar digital tersebut dipecah ke dalam *palette* berukuran 15x15 blok piksel, dengan algoritma sebagai berikut :

```
        pal = [x for x in range(256) if
    x%int(opt.impalred) == 0]
         if pal[-1] != 255:
             pal.append(255)
         imagetopalette(img, pal)
         pix = img.load()

         for x in range(w-block_len):
          for y in range(h-block_len):
           data = list(blockpoints(pix, (x,y),
    block_len)) + [(x,y)]
            parts.append(data)
         parts = sorted(parts)
return parts
```

### *3.4.3.2 Similarparts*

*Similarparts* pada dasarnya akan mencari dan menyimpan bagian piksel blok yang identik dari gambar digital yang telah dipecah pada proses sebelumnya, jika kedua bagian tersebut tidak identik maka algoritma akan melanjutkan ke bagian blok piksel berikutnya. Berikut algoritma untuk mencari blok piksel yang identik :

```
    dupl = []
     global opt
     l = len(imagparts[0])-1


     for i in range(len(imagparts)-1):
      difs = sum(abs(x-y) for x,y in
zip(imagparts[i][:l],imagparts[i+1][:l]))
      mean = float(sum(imagparts[i][:l])) / l
      dev = float(sum(abs(mean-val) for val in
imagparts[i][:l])) / l
      if dev/mean >= float(opt.blcoldev):
       if difs <= int(opt.blsim):
        if imagparts[i] not in dupl:
         dupl.append(imagparts[i])
        if imagparts[i+1] not in dupl:
         dupl.append(imagparts[i+1])
```

```
        return dupl
```

### *3.4.3.3 Clusterparts*

Pada bagian proses ini akan dipilih antara blok piksel yang memiliki kesamaan dan blok yang tidak memiliki kesamaan. Pemilihan blok piksel pada proses ini dilakukan dengan metode *"Hausdorff distance"* yang telah dimodifikasi dari aslinya berikut rumusan asli dari *"Hausdorff distance":*

$$d_{\mathrm{H}}(X,Y) = \max\{ \sup_{x\in X} \inf_{y\in Y} d(x,y), \sup_{y\in Y} \inf_{x\in X} d(x,y) \},$$

Karena metode *"Hausdorff distance"* ini pada awalnya dibuat untuk mencari jarak maksimal dua buah titik pada sebuah metriks, dirubah menjadi mencari jarak rata – rata. Berikut adalah algoritma dari *"Hausdorff distance"* yang telah dimodifikasi untuk mencari jarak rata – rata :

```
    if forward == None:
      return
max(Hausdorff_distance(clust1,clust2,True,dir),Haus
dorff_distance(clust1,clust2,False,dir))
     else:
      clstart, clend = (clust1,clust2) if forward
else (clust2,clust1)
      dx, dy = dir if forward else (-dir[0],-
dir[1])
      return sum([min([Dist((p1[0]+dx,p1[1]+dy),p2)
for p2 in clend]) for p1 in clstart])/len(clstart)
```

### *3.4.3.4 Marksimilar*

Proses terakhir yaitu menandai area yang teridentifikasi identik pada daerah gambar digital. Pada proses ini algoritma "memanggil" sebuah fungsi yang terdapat pada *library* PIL *(python image library)* yang bernama *"ImageDraw"*. Fungsi ini mampu memberikan tanda arsiran berwarna merah muda atau apa saja pada daerah yang telah teridentifikasi adanya kesamaan blok piksel. Berikut algoritma "pemanggilan" fungsi *ImageDraw :*

```
    global opt
     blocks = []
     if clust:
      draw = ImageDraw.Draw(image)
      mask = Image.new('RGB', (size,size), 'cyan')
      for cl in clust:
       for x,y in cl:
            im = image.crop((x,y,x+size,y+size))
            im = Image.blend(im,mask,0.5)
            blocks.append((x,y,im))
      for bl in blocks:
      x,y,im = bl
image.paste(im,(x,y,x+size,y+size
```