BAB 3

ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Masalah

Bangunan-bangunan berarsitektur tradisional di Indonesia banyak sekali, seperti halnya rumah-rumah atau bangunan adat yang ada di Indonesia. Antara lain, bangunan jawa, bali, sumatera, kalimanta, sulawesi dan papua. Dengan beranekaragaman bangunan adat dengan arsitektur bangunan yang berbeda-beda yang begitu banyak, kadang membuat kita sulit untuk membedakan sebuah arsitektur bangunan tradisional. Dikarenakan ada beberapa bangunan rumah tradisional Indonesia yang memiliki bagian yang sama dengan bangunan rumah tradisional yang lainnya dan bangunan rumah tradisonal diindonesia yang cukup banyak. Sehingga kadang kita sulit untuk membedakannya dan tidak mengetauhi nama bangunan rumah tradisional yang ada di Indonesia tersebut. Oleh karena itu, pengenalan arsitektur bangunan ini merupakan alternatif yang baik untuk dikembangkan karena bisa mempermudah untuk mengenali arsitektur bangunan rumah tradisional di Indonesia. Maka dibutuhkan sebuah aplikasi yang bisa membantu memecahkan masalah tersebut ialah bagaimana cara untuk mempermudah mengidentifikasi atau mengenali sebuah arsitektur bangunanrumah tradisonal Indonesia pada gambar.

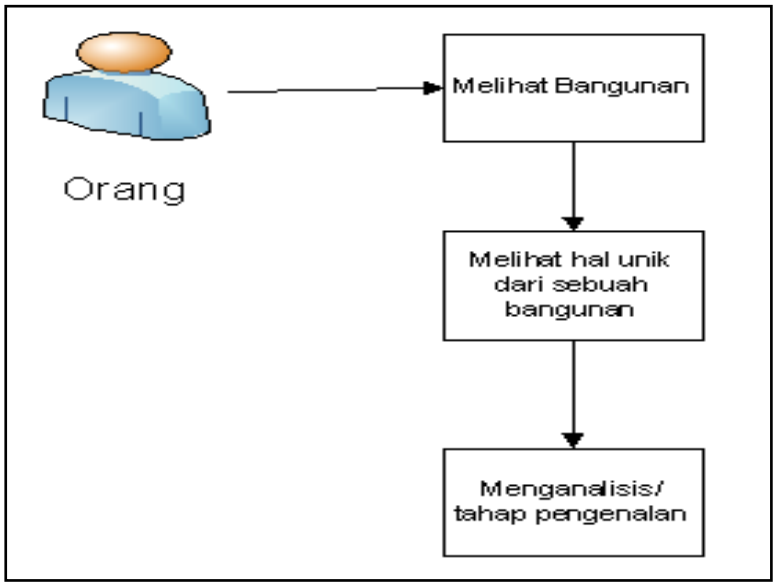
3.2 Analisis Sistem

Analisis sistem merupakan suatu cara yang digunakan dalam mengidentifikasi dan mengevaluasi suatu permasalahan, hambatan yang dihadapi dan kebutuhan yang diharapkan dalam sebuah sistem. Analisa ini diperlukan sebagai dasar untuk tahapan perancangan sistem. Hasil dari proses analisis sistem ini akan menghasilkan berbagai kesimpulan dan saran yang dapat digunakan

untuk dijadikan dasar perancangan aplikasi yang akan dibangun. Analisis sistem yang dilakukan untuk membangun aplikasi ini sebagai berikut :

3.2.1 Analisis Prosedur yang Sedang Berjalan

Berdasarkan hasil pengamatan, maka dapat diambil kesimpulan mengenai prosedur bagaimana seseorang bisa mengenali jenis arsitektur suatu bangunan. Penjelasannya sebagai berikut :



Gambar 3. 1 Analisis Prosedur yang Sedang Berjalan

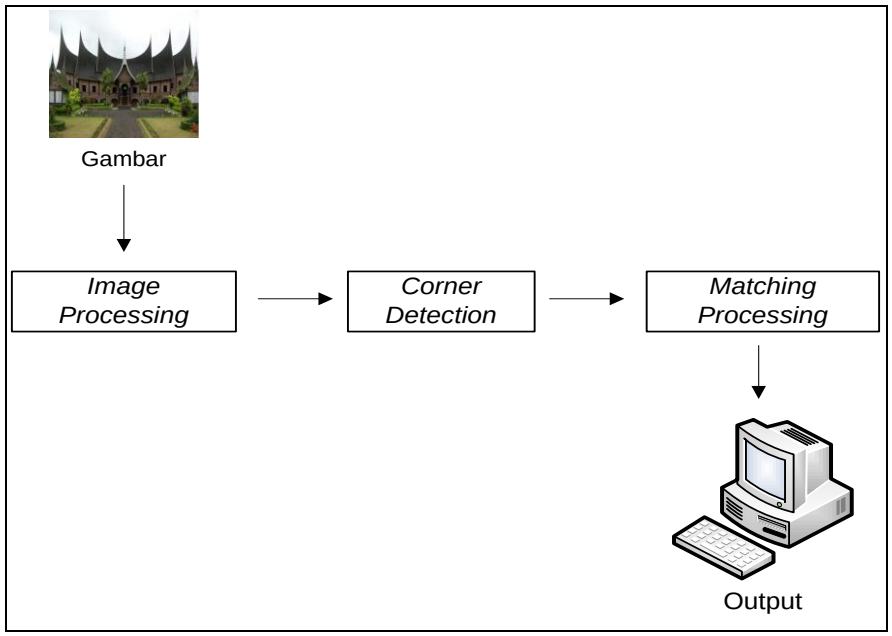
Adapun penjelasan dari gambar diatas sebagai berikut :

1. Orang melihat sebuah bangunan
2. Melihat sesuatu yang unik atau sesuatu yang menjadi sebuah ciri khas yang ada bangunan tersebut (dilihat dari bentuk fisik keseluruhan bangunan, bentuk atap, bentuk ornamen bangunan, dll).
3. Mengenali atau mengkira-kira arsitektur bangunan tersebut merupakan termasuk arsitektur bangunan dari salah satu daerah. (pada tahap mengenali arsitektur bangunan ini pengetahuan jenis arsitektur bangunan tradisional di Indonesia atau rumah-rumah adat yang dimiliki seseorang sangat berperan penting. Karena semakin banyak pengetahuan seseorang tentang arsitektur

tradsional bangunan di Indonesia semakin meminimalkan kesalahan persepsi tentang pengenalan bangunan tersebut)

3.2.2. Analisis Sistem yang Akan Dibangun

Sistem yang akan di bangun adalah sistem pengidentifikasian atau pengenalan jenis arsitektur bangunan tradisional yang ada di Indonesia. Sistem ini bisa mengenali suatu arsitektur bangunan di sebuah citra. Adapun gambaran umum tentang sistem yang akan dibangun dapat dilihat pada Gambar 3.2.



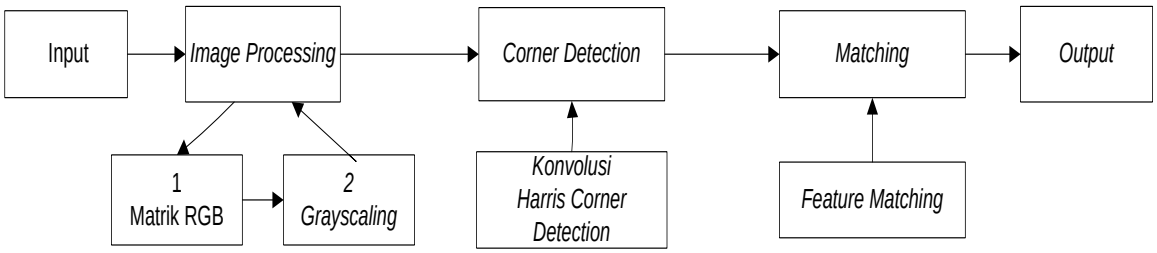
Gambar 3. 2 Alur Proses Kerja Sistem yang Akan Dibangun

Gambar 3.2 menggambarkan bahwa data yang dipilh berupa gambar bangunan tradisonal indonesia. Terdapat proses untuk mengenali sebuah arsitektur bangunan tradisional yaitu image processing, corner detection dan matching processing. Hasil dari setiap proses ditampilkan pada layar komputer.

3.3 Analisis Metode

Analisis metode digunakan untuk mengetahui alur proses dari sebuah metode yang digunakan dapat diterapkan ke dalam aplikasi yang dibangun. Pembangunan aplikasi identifikasi arsitektur bangunan ini menggunakan metode *Harris Corner Detection* pada *Feature Matching* untuk mencocokan tiap-tiap

bagian dari suatu citra dengan citra latih. Sehingga mendapatkan kondisi *match* dan bisa diindetitikasi arsitektur bangunan pada citra. Adapun penerapan metode pada alur kerja sistem dapat dilihat pada gambar 3.3.



Gambar 3. 3 Penerapan Metode Pada Alur Kerja Sistem

Berikut alur kerja sistem yang akan dibangun adalah :

1. Input, Pengguna mengambil gambar atau memilih cirta/gambar bangunan tradisional yang akan di identifikasi pada data yang ada dikomputer
2. Tahap *image processing* dimana pada tahap ini citra yang dipilih pengguna untuk dikenali akan dikonversi ke bentuk matrik RGB, setalah menjadi matrik RGB akan diubah menjadi citra *grayscale* atau citra keabuan.
3. Suatu citra warna di ubah menjadi citra *grayscale* . untuk memperoleh informasi intensitas dari gambar, sehingga dapat di sortir mulai dari hitam untuk intensitas yang paling lemah samapai dengan putih yang intensitas yang paling kuat. Dan mendapatkan nilai *grayscale* nya
4. *Corner detection*, pada tahap ini sistem mendeteksi *corner* (sudut) yang ada pada citra. Pada tahap ini metode *harris corner detection* dilakukan yang didalamyan terdapat proses konvolusi matrik piksel suatu citra. Tahap konvolusi dilakukan bersamaan pada saat melakukan metode *hariss corner detection*. Hasil yang didapatkan yaitu matrik nilai hasil konvolusi *hariss corner detection* dan pola *corner* pada citra.
5. Proses *matching*, dalam tahapan ini *feature matching* yang digunakan ialah *chamfer maching* yaitu melakukan pencocokan atas dasar menghitung jarak kesamaan antara dua buah gambar. setelah mendapatkan pola *corner* tahap selanjutnya ialah proses *Chamfer matching*. Citra yang sudah dideteksi

*corner*nya dan mendapatkan pola *corne*r nya. Akan dihitung jarak cornernya dengan citra latih. Dengan cara menumpuk pola corner citra latih ke pola *corner* citra uji. Sehingga mendapatkan nilai jarak persamaan anatar kedua gambar tersebut. Semakin rendah nilai kesamaan jarak semakin baik hasilnya.

6. Outputnya, menampilkan gambar yang dipilih oleh pengguna serta menampilkan gambar hasil pengenalan yang memiliki nilai jarak kesamaan yang terendah dengan citra uji

Untuk lebih jelas mengenai proses-proses analisis yang dilakukan pada sistem ini, penjelasanya sebagai berikut :

3.3.1 Analisis Matrik RGB (*Red, Green, Blue*)

RGB merupakan suatu model warna yang terdiri dari 3 buah warna yaitu R(*Red*), G(*Green*), dan B(*Blue*). Tahapan awal yaitu mengkonversi *pixel-pixel* matrik akan di pisah-pisah lagi menjadi matrik RGB. Setelah menjadi matrik RGB akan di konversi ke masing-masing matrik diantaranya matrik R, matrik G, dan matrik B. Gambarannya bisa dilihat pada gambar 3.4.

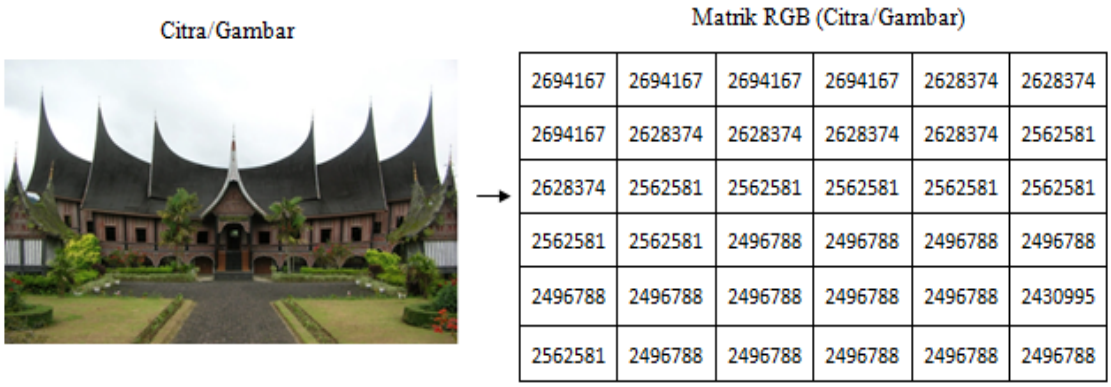
Pixel pada Matrik

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

→

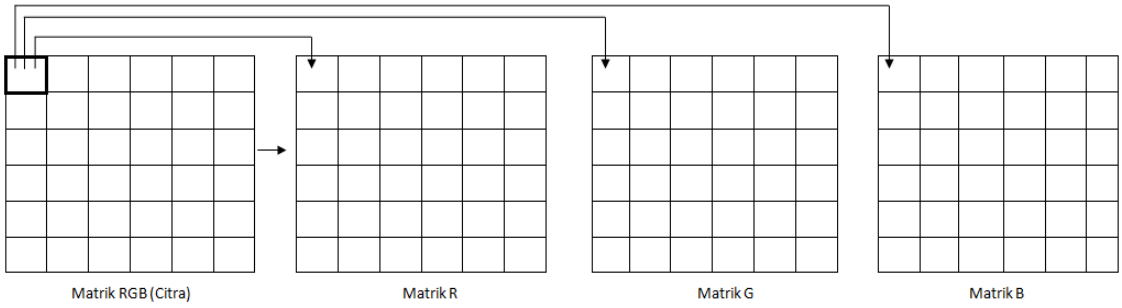
Matrik Citra/Gambar

2694167	2694167	2694167	2694167	2628374	2628374
2694167	2628374	2628374	2628374	2628374	2562581
2628374	2562581	2562581	2562581	2562581	2562581
2562581	2562581	2496788	2496788	2496788	2496788
2496788	2496788	2496788	2496788	2496788	2430995
2562581	2496788	2496788	2496788	2496788	2496788



Gambar 3. 4 Konversi Citra Menjadi Matrik RGB

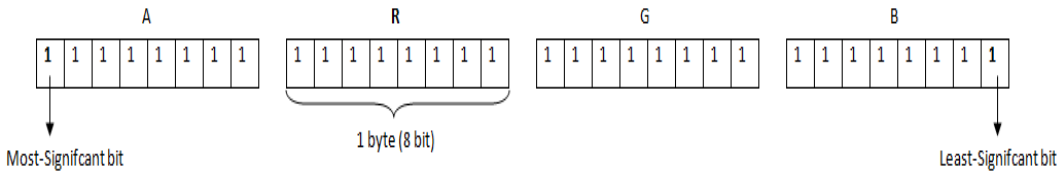
Pengkonversian matrik RGB menjadi masing-masing matrik R, G dan B dapat digambarkan sebagai berikut :



Gambar 3. 5 Konversi Matrik RGB ke Matrik (R, G & B)

Nilai dari 1 piksel yang ada matrik RGB akan di konversi ke nilai piksel ke matrik R, matrik G, dan matrik B. Jumlah kolom dan baris harus sama dan apabila pada posisi *pixel* (0,0) pada image maka pada matrik R, G, dan B pun mesti pada posisi *pixel* (0,0).

Biasanya untuk mendapatkan nilai RGB dengan cara *bit shifting* (penggeseran bit). Setiap *pixel* hanya memiliki 1 nilai yaitu 32bit yang bertipe integer. Gambarannya bisa dilihat pada gambar 3.6 berikut :



Gambar 3. 6 Formula ARGB 32-bit

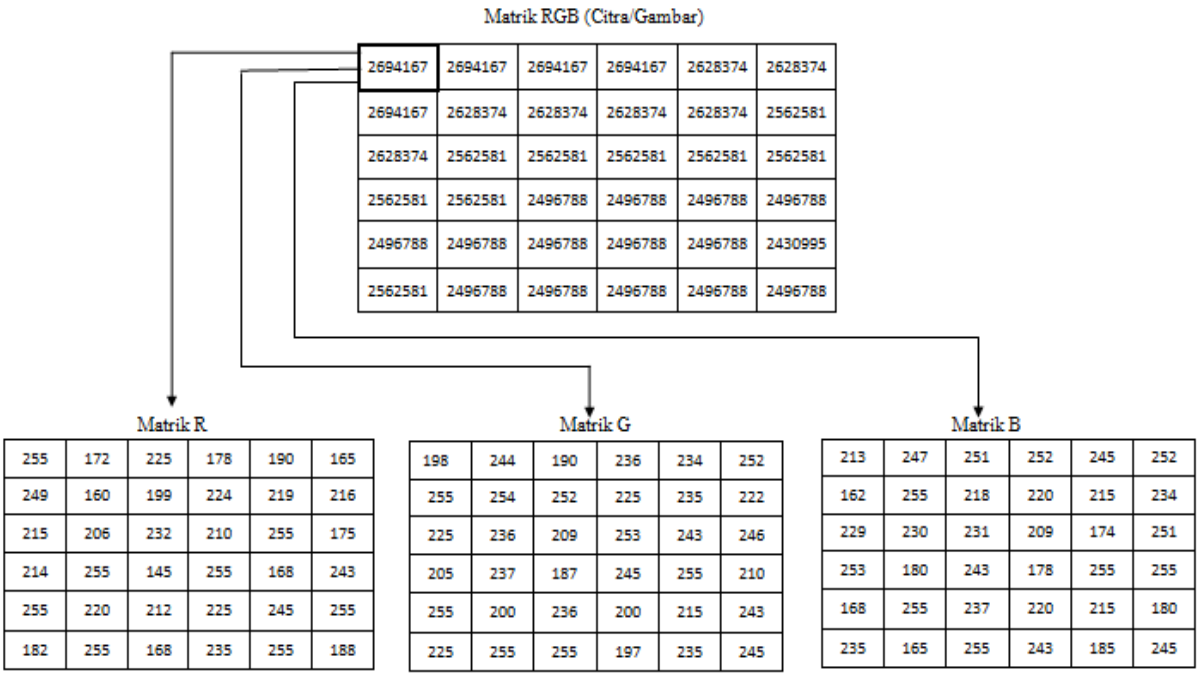
Setiap warna memiliki 1byte (8 bit) dimana nilainya terdiri dari 256 *color* yang range nya (0 - 255). Nilai RGB didapat dengan cara menggeser dan di-& kan. Adapun proses penggeseran bit (*bit shifting*) dapat dilakukan dengan ketentuan berikut :

R = (px >> 16bit) &255

G = (px >> 8bit) &255

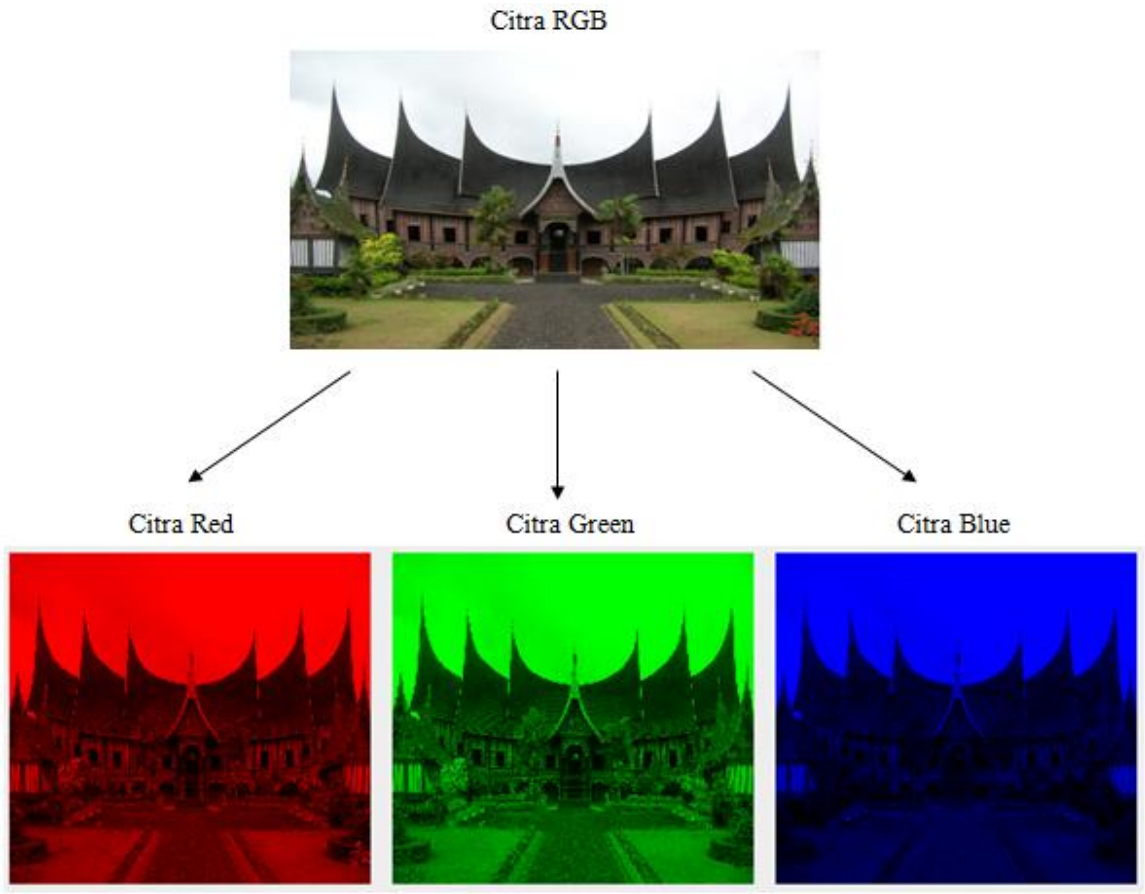
B = (px & 255)

Berikut ini gambaran konversi matrik citra (matrik RGB) ke masing-masing matrik RGB (matrik R, G dan B). Citra berukuran 6 x 6. Adapun hasilnya pengkonversiannya bisa dilihat pada gambar 3.7. dan gambar 3.8



Gambar 3. 7 Hasil Konversi Matrik RGB Menjadi Matrik (R, G dan B)

Berikut ini citra hasil *corner detection*



Gambar 3. 8 Citra *Red, Green* dan *Blue*

Algoritma untuk mendapatkan gambar *Red*, *Green* dan *Blue* adalah sebagai berikut :

```
(* untuk mendapatkan matrik red, green dan blue dengan cara penggeseran)
     Red =(rgb >> 16) & 255;
     Green =(rgb >> 8) & 255;
     Blue = (rgb & 0xff);
     Alpha =(rgb >> 24) & 255;

Procedure getChanelRGB()

Kamus
   Rgb, height, width : int

Algoritma
        for (int i = 0; i < height; i++)
            for (int j = 0; j < width; j++)
                    CHANNEL_RED:
                       result[i][j] = Red(rgb[i][j]);
                    else
                    CHANNEL_GREEN:
```

```
                    result[i][j] = Green(rgb[i][j]);
                else
                CHANNEL_BLUE:
                    result[i][j] = getBlue(rgb[i][j]);
            endfor
        endfor
```

3.3.2 Analisis Tahap *Grayscaling*

Tahap selanjutnya adalah tahap *grayscaling* yaitu mengubah citra warna menjadi citra grayscale, hal ini digunakan untuk menyederhanakan model citra. Untuk memperoleh informasi intensitas dari citra tersebut, citra dapat di sortir secara ekslusif mulai dari hitam untuk intensitas yang paling lemah sampai dengan putih untuk intensitas yang paling kuat. Citra warna terdiri dari 3 layer matrik yaitu R-layer, G-layer, B-layer. Jadi dalam proses ini akan mengubah 3 layer matriks citra berwarna menjadi 1 layer matriks *grayscale* dan hasilnya citra *grayscale*. Dalam citra ini tidak ada lagi warna, yang ada adalah derajat keabuan

Untuk mengubah citra berwarna yang mempunyai nilai matrik masing-masing R, G dan B menjadi citra *grayscale* dengan nilai *S*. Dengan menghitung nilai rata-rata RGB dari nilai R, G dan B maka konversi RGB ke *grayscale* dapat dilakukan. Berikut adalah perhitungan untuk mendapatkan *grayscale* :

$$S = \frac{\mathrm{R} + \mathrm{G} + \mathrm{B}}{3} \quad \text{.....(3.1)}$$

Dimana :

S = array (i,j) dari matrik *grayscale*

R = nilai *Red* dari *pixel* (i,j) pada citra

G = nilai *Green* dari *pixel* (i,j) pada citra

B = nilai *Blue* dari *pixel* (i,j) pada citra

Misalkan suatu citra bangunan pada satu *pixel* (0,0) memiliki nilai

R = 255 G = 198 B = 213

Berikut adalah hasil perhitungan *grayscal*e :

$$s = \frac{255 + 198 + 213}{3}$$

$$s = \frac{666}{3}$$

$$s = 222$$

Jadi rata-rata *grayscale* dari citra bangunan pada *pixel* (0,0) adalah 222

Untuk mendapatkan keseluruhan nilai *pixel grayscale* pada citra dilakukan perhitungan yang sudah dijelaskan sebelumnya. Adapun perubahan citra RGB (citra awal) menjadi citra *grayscale* dapat dilihat pada gambar 3.9.

Matrik RGB (Citra/Gambar)

2694167	2694167	2694167	2694167	2628374	2628374
2694167	2628374	2628374	2628374	2628374	2562581
2628374	2562581	2562581	2562581	2562581	2562581
2562581	2562581	2496788	2496788	2496788	2496788
2496788	2496788	2496788	2496788	2496788	2430995
2562581	2496788	2496788	2496788	2496788	2496788

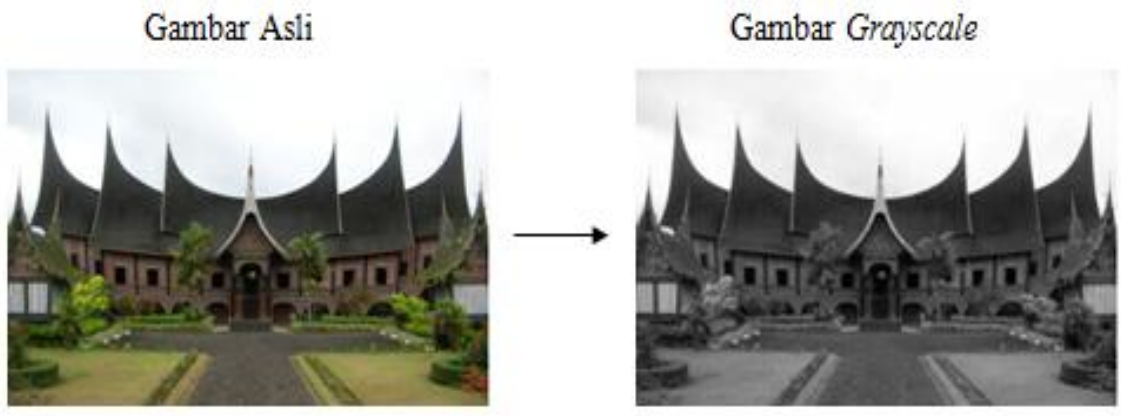
→

Matrik *Grayscale*

222	222	222	222	223	223
222	223	223	223	223	224
223	224	224	224	224	224
224	224	225	225	225	225
225	225	225	225	225	226
224	225	225	225	225	225

Gambar 3. 9 Matrik RGB (Citra Awal) Menjadi Matrik *Grayscale*

Berikut ini adalah citra hasil *grayscaling*



Gambar 3. 10 Citra RGB (Citra Awal) Menjadi Citra *Grayscale*

Algoritma untuk proses *grayscale* gambar adalah sebagai berikut :

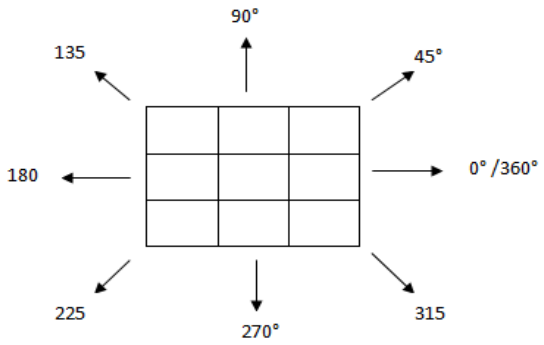
```
Procedure getRGB()

Kamus
   I, j, width, height : int

Algoritma
for i ← 0 to width do
    for j ← 0 to height do
        pixelgray ← (gambar[i][j].R + gambar[i][j].G + gambar[i][j].B) / 3
         gambar[i][j].R ← pixelgray
         gambar[i][j].G ← pixelgray
         gambar[i][j].B ← pixelgray
    endfor
endfor
```

3.3.3 Analisis Metode *Harris Corner Detection*

Untuk menghasilkan nilai konsistensi citra yang tidak memperdulikan adanya rotasi, skala, variasi pencahayaan maupun *noise* pada *featur matching* yaitu menggunakan *harris corner detection*. Pada tahap ini proses konvolusi dilakukan akan tetapi pada kasus ini tidak seperti konvolusi biasanya yang menempelkan kernel pada sudut kiri dan kanan saja. Akan tetapi dengan adanya metode *harris corner* konvolusi dilakukan pada suatu titik sudut dengan menggeser kernel (*window)* tersebut kesegalah arah sehingga didapatkan nilai perbedaan yang besar. Adapun penggeseran kernel ke segalah arah bisa dilihat pada gambar 3.11 berikut :



Gambar 3. 11 Pergeseran Window ke Beberapa Arah (Dalam Derajat)

Adapun penggeseran kernel (*shift*) yang dilakukan pada citra dengan metode *harris corner detection* sebagai berikut :

1. Kondisi Awal

Matrik Citra *Grayscale*

222	222	222	222	223	223
222	223	223	223	223	224
223	224	224	224	224	224
224	224	225	225	225	225
225	225	225	225	225	226
224	225	225	225	225	225

Pada *pixel* (0,0)

(a) Penggeseran Kernel (*window*) pada *pixel* (0,0)

- Kondisi Awal pada *pixel* (0,0)

Pada *pixel* (0,0)

	222	222	222	222	223	223
	222	223	223	223	223	224
	223	224	224	224	224	224
	224	224	225	225	225	225
	225	225	225	225	225	226
	224	225	225	225	225	225

Kernel pada *pixel* (0,0)

	222	222
	222	223

  - Kernel digeser 0°

222	222	222	222	223	223
222	223	223	223	223	224
223	224	224	224	224	224
224	224	225	225	225	225
225	225	225	225	225	226
224	225	225	225	225	225

Kernel hasil penggeseran 0°

222	222	222
222	223	223

- Kernel digeser 45°

222	223	223	223	223	224
223	224	224	224	224	224
224	224	225	225	225	225
225	225	225	225	225	226
224	225	225	225	225	225
222	223	223	223	223	224

Kernel hasil penggeseran 45°

222	223	223

- Kernel digeser 90°

	222	223	223	223	223	224
	223	224	224	224	224	224
	224	224	225	225	225	225
	225	225	225	225	225	226
	224	225	225	225	225	225
	222	223	223	223	223	224

Kernel hasil penggeseran 90°

	222	223

- Kernel digeser 135°

		222	222	222	222	223	223
		222	223	223	223	223	224
		223	224	224	224	224	224
		224	224	225	225	225	225
		225	225	225	225	225	226
		224	225	225	225	225	225

Kernel hasil penggeseran 135°

		222

- Kernel digeser 180°

		222	222	222	222	223	223
		222	223	223	223	223	224
		223	224	224	224	224	224
		224	224	225	225	225	225
		225	225	225	225	225	226
		224	225	225	225	225	225

Kernel hasil penggeseran 180°

222		
222		

- Kernel digeser 225°

		222	222	222	222	223	223
		222	223	223	223	223	224
		223	224	224	224	224	224
		224	224	225	225	225	225
		225	225	225	225	225	226
		224	225	225	225	225	225

Kernel hasil penggeseran 255°

		222
		222
		223

- Kernel digeser 270°

	222	222	222	222	223	223
	222	223	223	223	223	224
	223	224	224	224	224	224
	224	224	225	225	225	225
	225	225	225	225	225	226
	224	225	225	225	225	225

Kernel hasil penggeseran 270°

	222	222
	222	223
	223	224

- Kernel digeser 315°

222	222	222	222	223	223
222	223	223	223	223	224
223	224	224	224	224	224
224	224	225	225	225	225
225	225	225	225	225	226
224	225	225	225	225	225

Kernel hasil penggeseran 315°

222	222	222
222	223	223
223	224	224

- Kernel digeser 360°

  Apabila kernel digeser 360°, maka kernel kembali ke posisi 0°. Sehingga untuk perhitungannya nanti posisi 0° dan 360° merupakan satu nilai.

Algoritma Proses penggeseran ke segalah arah adalah sebagai berikut :

```
procedur getCoocurrence(matrik, index : int)

Kamus
Matrik, width, height, degree, i, j, index : int
iIndex, jIndex : int

Algoritma
     cooccurrence = new [NUMBER_OF_GRAY][NUMBER_OF_GRAY];
               DEGREE_0:
                for (i = 0; i < height; i++)
                    for (j = 0; j < width - 1; j++)
                        iIndex = Index(matrix[i][j]);
                        jIndex = Index(matrix[i][j + 1]);
                        cooccurrence[iIndex][jIndex]++;
                    endfor
                endfor

            DEGREE_45:
                for (i = 1; i < height; i++)
                    for (j = 0; j < width - 1; j++)
                        iIndex = Index(matrix[i][j]);
```

```
                jIndex = Index(matrix[i + 1][j + 1]);
                cooccurrence[iIndex][jIndex]++;
            endfor
        endfor

    DEGREE_90:
        for (i = 1; i < height; i++)
            for (j = 0; j < width; j++)
                iIndex = Index(matrix[i][j]);
                jIndex = Index(matrix[i - 1][j]);
                cooccurrence[iIndex][jIndex]++;
            endfor
        endfor

    DEGREE_135:
        for (i = 1; i < height; i++)
            for (j = 1; j < width; j++)
                iIndex = Index(matrix[i][j]);
                jIndex = Index(matrix[i - 1][j - 1]);
                cooccurrence[iIndex][jIndex]++;
            endfor
        endfor

    DEGREE_180:
        for (i = 1; i < height; i++)
            for (j = 1; j < width; j++)
                iIndex = Index(matrix[i][j]);
                jIndex = Index(matrix[i - 1][j - 1]);
                cooccurrence[iIndex][jIndex]++;
            endfor
        endfor

    DEGREE_225:
        for (i = 1; i < height; i++)
            for (j = 1; j < width; j++)
                iIndex = Index(matrix[i][j]);
                jIndex = Index(matrix[i - 1][j - 1]);
                cooccurrence[iIndex][jIndex]++;
            endfor
```

```
            endfor

        DEGREE_270:
            for (i = 1; i < height; i++)
                for (j = 1; j < width; j++)
                    iIndex = Index(matrix[i][j]);
                    jIndex = Index(matrix[i - 1][j - 1]);
                    cooccurrence[iIndex][jIndex]++;
                endfor
            endfor

        DEGREE_315:
            for (i = 1; i < height; i++)
                for (j = 1; j < width; j++)
                    iIndex = Index(matrix[i][j]);
                    jIndex = Index(matrix[i - 1][j - 1]);
                    cooccurrence[iIndex][jIndex]++;
                endfor
            endfor
```

Didalam metode *harris corner detection* untuk mendapatkan nilai intensitas bisa dengan menggunakan rumus 3.2 sebagai berikut :

$$E(u,v) = \sum_{x,y} w\,(x,y)[I(x+u,y+v) - I(x,y)]^2 \dots\dots\dots\dots (3.2)$$

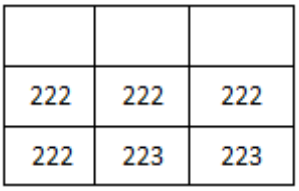
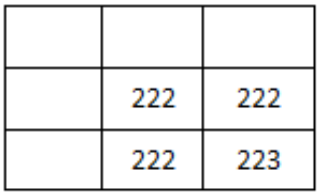
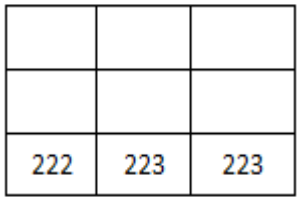
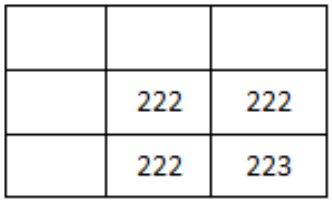
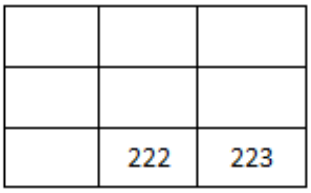
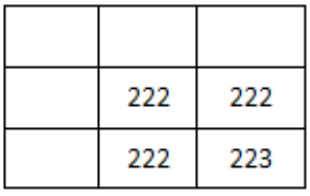
Dimana :

- $\sum_{x,y} w\,(x,y)$: Window Function
- $I(x+u,y+v)$: Insensity Shifted
- $I(x,y)$: Intensity

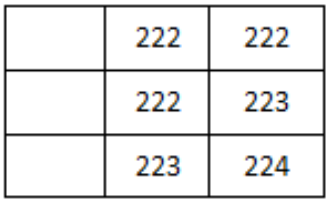
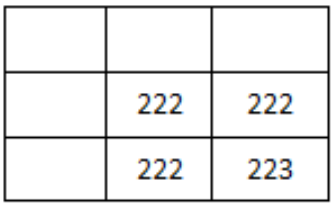
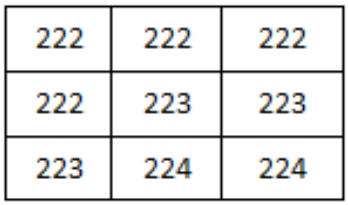
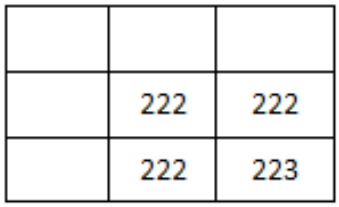
Menurut paper Cucun Very A dan Darya Frolova, Denis Simakov pada slide "*Matching with Invariant Features*", The Weizmann Insitute of Science. Bawasanya fungsi windowing $\sum_{x,y} w\,(x,y)$ bernilai 1 jika didalam jendela window atau bernilai 0 jika diluar jendelan window.

Perhitungan nilai intensitas pada citra dapat dilihat pada tabel 3.1 berikut :

Tabel 3. 1 Perhitungan Nilai Intensitas

Posisi Penggeseran	**S** **(kernel hasil penggeseran)**	**T** **(kernel awal pada posisi *pixel* (0,0))**
0°/360°		
Perhitungan	= 1 [(222-0)+(222-222)+(222-222)+(222-0)+(223-222)+(223-223)]² = 1 [(222)+(222)+(1)] ² = 1 [445]² = 198025	
45°		
Perhitungan	= 0 [(0-222)+(0-222)+(0-222)+(222-0)+(223-222)+(223-223)]² = 0 [(-222)+(-222)+ (-222)+(222)+(1)] ² = 0 [889]² = 0	
90°		
Perhitungan	= 0 [(0-222)+(0-222)+(222-222)+(223-223)]² = 0 [(-222)+(-222)] ² = 0 [444]²	

<table>
<tr><td></td><td colspan="2">= 0</td></tr>
<tr><td>135°</td><td><table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td>222</td></tr></table></td><td><table><tr><td></td><td></td><td></td></tr><tr><td></td><td>222</td><td>222</td></tr><tr><td></td><td>222</td><td>223</td></tr></table></td></tr>
<tr><td>Perhitungan</td><td colspan="2">= 0 [(0-222)+(0-222)+(0-222)+(222-223)]²
= 0 [(-222)+(-222)+ (-222)+(-1)] ²
= 0 [667]²
= 0</td></tr>
<tr><td>180°</td><td><table><tr><td></td><td></td><td></td></tr><tr><td>222</td><td></td><td></td></tr><tr><td>222</td><td></td><td></td></tr></table></td><td><table><tr><td></td><td></td><td></td></tr><tr><td></td><td>222</td><td>222</td></tr><tr><td></td><td>222</td><td>223</td></tr></table></td></tr>
<tr><td>Perhitungan</td><td colspan="2">= 0 [(222-0)+(0-222)+(0-222)+(222-0)+(0-222)+((0-223)]²
= 0 [(222)+(-222)+ (-222)+(222)+ (-222)+(-223)] ²
= 0 [1333]²
= 0</td></tr>
<tr><td>225°</td><td><table><tr><td></td><td></td><td>222</td></tr><tr><td></td><td></td><td>222</td></tr><tr><td></td><td></td><td>223</td></tr></table></td><td><table><tr><td></td><td></td><td></td></tr><tr><td></td><td>222</td><td>222</td></tr><tr><td></td><td>222</td><td>223</td></tr></table></td></tr>
<tr><td>Perhitungan</td><td colspan="2">= 0 [(222-0)+(0-222)+(222-222)+(0-222)+(223-223)]²
= 0 [(222)+(-222)+ (-222)] ²
= 0 [666]²
= 0</td></tr>
</table>

<table>
<tr><td>270°</td><td></td><td></td></tr>
<tr><td>Perhitungan</td><td colspan="2">= 1 [(222-0)+(0-222)+(222-222)+(223-223)+(223-222)+(224+223)]²
= 1 [(222)+(-222)+ (1)+(1)+(1)] ²
= 1 [445]²
= 198025</td></tr>
<tr><td>315°</td><td></td><td></td></tr>
<tr><td>Perhitungan</td><td colspan="2">= 1 [(222-0)+(222-0)+(222-0)+(222-0)+(222-222)+(223-222)+(223-0)+(224- 222)+(224+223)]²
= 1 [(222)+(222)+(222)+(222)+ (1)+(223)+(2)+(1)] ²
= 1 [1115]²
= 1243225</td></tr>
<tr><td colspan="3">$\sum(0,0) = 198025 + 198025 + 1243225$
$\sum(0,0) = 1639275$</td></tr>
</table>

Dari tabel 3.1. didapatkan jumlah nilai intensitas (*frame diferences*) pada *pixel* (0,0) adalah 1639275. Untuk mendapatkan jumlah semua nilai intensitas (*frame diferences*) setiap *pixel* pada citra dilakukan Proses perhitungan yang sudah dijelaskan tersebut (penggeseran kernel (*shift*) kesegalah arah dan perhitungan nilai intensitas seperti tabel 3.1.). Sehingga didapatkan matrik nilai intensitas (*frame diferences*) pada citra sebagai berikut :

Pixel pada Matrik (x,y)

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

→

Matrik Nilai Intensitas

1639275	1752352	2157735	2357651	1637619	1976751
2163645	2056765	1663255	2232462	1834862	2060532
1566156	1875651	1976654	1550678	2563225	2253216
2346731	1950601	2123676	1863042	2463216	1876349
1637671	2354756	2105943	1956786	1786326	2347690
1719826	2324567	1697623	2136432	1636833	1637725

Menurut paper Cucun Very A dan Darya Frolova, Denis Simakov pada slide "*Matching with Invariant Features*", The Weizmann Insitute of Science. Apabila nilai pada bagian *Intensity Shifted* dan *Intensity* untuk bagian yang flat/konstan akan bernilai kecil, sedangkan untuk daerah yang mengandung corner akan bernilai besar. Selain itu dikenali sebagai *corner* apabila nilai respon *corner* (*R*) lebih besar (>) dari pada threshold (R>*threshold*). Berdasarkan penelitian yang sudah ada yang dilakukan oleh Yuan-Liang Tang [18] bawasanya nilai *threshold* nya adalah 5000000 akan tetapi pada aplikasi ini nilai *threshold* 2000000. Dikarenakan melihat nilai matrik intensitas yang ada tidak nilai pada pixel yang melebihi nilai 5000000.

Berikut ini citra hasil *corner detection*

Citra/Gambar Awal

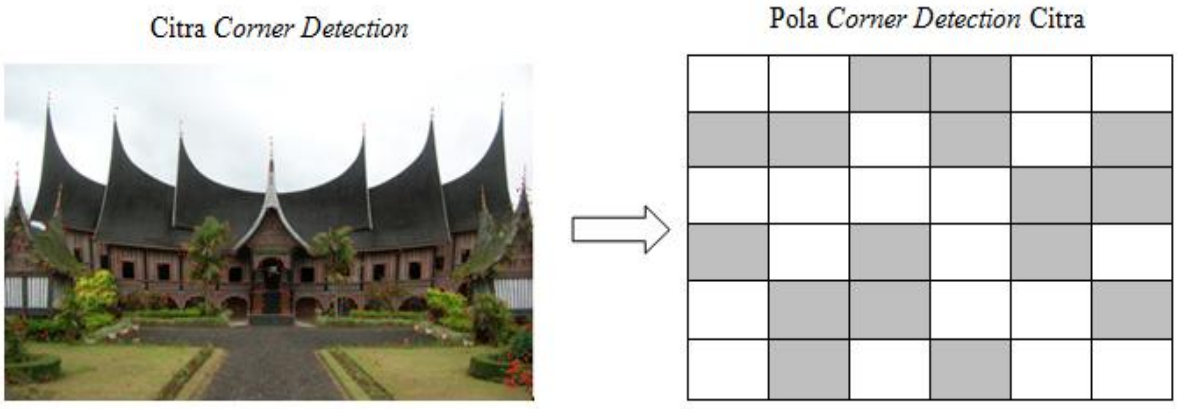


Citra/Gambar Hasil *Corner Detection*



Gambar 3. 12 Citra Hasil *Corner Detection*

Dari citra yang terdeteksi *corner* nya maka bisa didapatkan pola *corner detection* pada citra tersebut. *Pixel* yang terdapat *corner* diberi *block* warna abu. Dapat dilihat pada gambar 3.13



Gambar 3. 13 Pola *Corner Detection* Citra

Algoritma untuk mendapatkan *corner* pada gambar adalah sebagai berikut :

```
Procedure getCorner ()

Kamus
   Matrik, height, width, threshold, i, j : integer
   kernel, intensityDifference  : integer
   corner : array [height][width] of boolean
Algoritma
        corner=new boolean[height][width];
        for(int i=1; i<height-1; i++)
            for(int j=1; j<width-1; j++)
                kernel=new int[3][3];
                kernel[0][0]=matrix[i-1][j-1];
                kernel[0][1]=matrix[i-1][j];
                kernel[0][2]=matrix[i-1][j+1];
                kernel[1][0]=matrix[i][j-1];
                kernel[1][1]=matrix[i][j];
                kernel[1][2]=matrix[i][j+1];
                kernel[2][0]=matrix[i+1][j-1];
                kernel[2][1]=matrix[i+1][j];
                kernel[2][2]=matrix[i+1][j+1];
```

```
            else
             intensityDifference=0;
             for(k=0; k<3; k++)
                 for(l=0; l<3; l++)
                     intensityDifference= (kernel[k][l]-matrix[i][j], 2);
                 endfor
             endfor
             else
             if(intensityDifference>threshold){
                 corner[i][j]=true;
             endif
        endfor
    return corner;
    endfor
```

Setiap citra baik citra uji atau citra latih dilakukan tahapan pengenalan seperti tahapan-tahapan yang sudah dipaparkan sebelumnya sampai untuk mendapatkan pola *corner detection* masing-masing citra. pada kasus ini memiliki 10 *class*, yang disetiap class nya memiliki label (pura besakih, rumah joglo, rumah honai, rumah betang, rumah tongkonan, rumah gadang, rumah bolon, rumah musalaki, rumah mamuju dan rumah banjar bubungan tinggi). Setiap *class* terdiri dari 3 data latih. Adapun gambar latih bisa dilihat pada gambar 3.15 merupakan pola *corner* gambar latih 1 (rumah tongkonan), gambar 3.16 merupakan pola *corner* gambar latih 2 (rumah bolon), dan gambar 3.17 merupakan pola *corner* gambar latih 3 (rumah rumah gadang)

3.3.4 Analisis *Feature Matching*

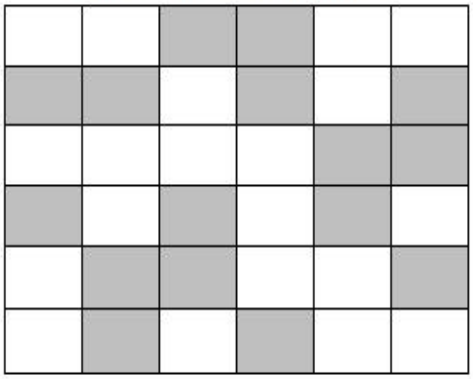
Tahap terakhir dalam pengenalan citra setelah proses diatas semua dilakukan yaitu proses *featur matching* dimana *feature matching* sendiri ialah mencocokan *corner* yang terdeteksi dan nilai intensitas perubahannya lebih besar dianggap *corner*. *Feature matching* yang digunakan dalam masalah ini ialah *chamfer matching. Chamfer matching* melakukan pencocokan atas dasar menghitung jarak (dis-kesamaan) antara dua buah gambar. awalnya ambil satu tititk/*pixel* dari kontur di citra uji dan menemukan jarak dari titik terdekat atau

pixel kontur digambar latih, jumlahkan jarak untuk semua titik/*pixel* gambar. ini semua untuk memberikan jarak kesamaan antara dua buah gambar. semakin rendah nilai semakin baik hasilnya. Adapun pola *corner detection* masing-masing citra dapat dilihat pada gambar 3.14.

Citra *Corner Detection*



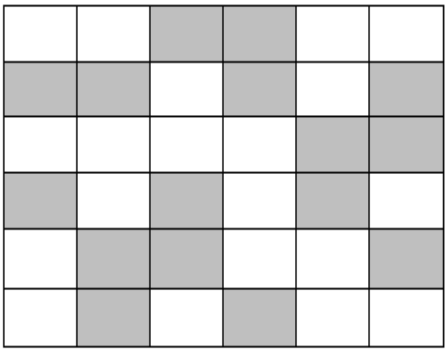
Pola *Corner Detection* Citra



Matrik Intensitas Citra Uji

1639275	1752352	2157735	2357651	1637619	1976751
2163645	2056765	1663255	2232462	1834862	2060532
1566156	1875651	1976654	1550678	2563225	2253216
2346731	1950601	2123676	1863042	2463216	1876349
1637671	2354756	2105943	1956786	1786326	2347690
1719826	2324567	1697623	2136432	1636833	1637725

Pola *Corner Detection* Citra Uji



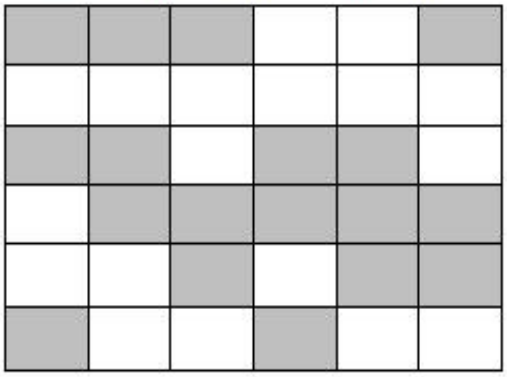
Gambar 3. 14 Pola *Corner Detection* Citra Uji

Pola *corner detection* pada citra latih 1

Citra Latih 1



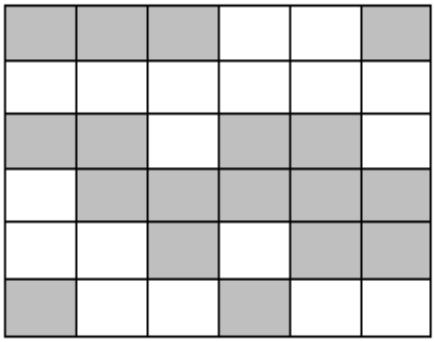
Pola *Corner Detection* Citra Latih 1



Matrik Intensitas Citra Latih 1

2456373	2051612	2167563	1756936	1899705	2056768
1456715	1976055	1576761	1886321	1976754	1456761
2123456	2025656	1889653	2356682	2056561	1987676
1564332	2234561	2066521	2297613	2656155	2231562
1674561	1986752	2356756	1987653	2565167	2345675
2456731	1478634	1978769	2055675	1656783	1767891

Pola *Corner Detection* Citra Latih 1



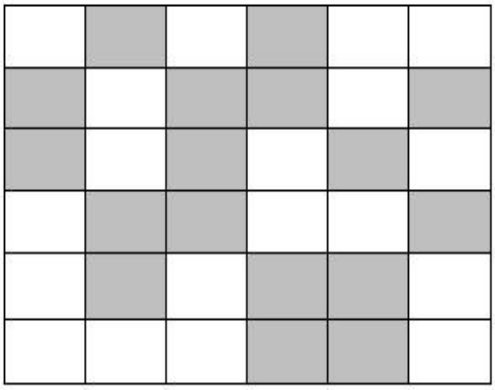
Gambar 3. 15 Pola *Corner Detection* Citra Latih 1

Pola *corner detection* pada citra latih 2

Citra Latih 2



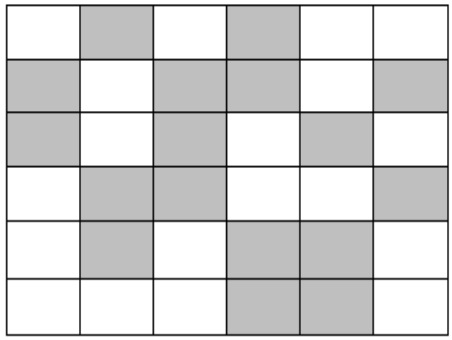
Pola *Corner Detection* Citra Latih 2



Matrik Intensitas Citra Latih 2

1678635	2158352	1797613	2467890	1765493	1650582
2567616	1976345	2656781	2234512	1836216	2566745
2253691	1876342	2063721	1997681	2024276	1798632
15636431	2456351	2136876	1688673	1456773	2178315
1876365	2124873	1876549	2356499	2458974	1983845
1976761	1456791	1976971	2011563	2133697	1543156

Pola *Corner Detection* Citra Latih 2



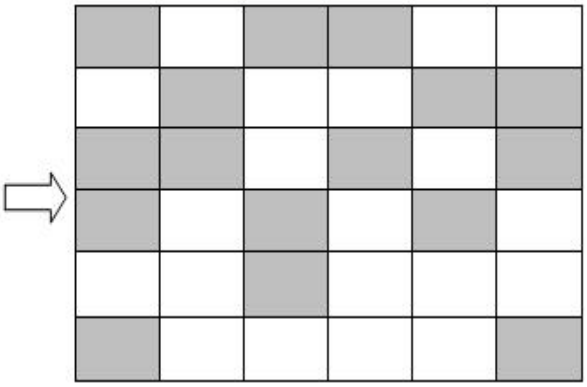
Gambar 3. 16 Pola *Corner Detection* Citra Latih 2

Pola *corner detection* pada citra latih 3

Citra Latih 3



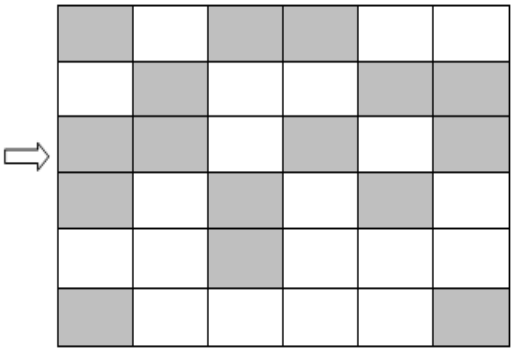
Pola *Corner Detection* Citra Latih 3



Matrik Intensitas Citra Latih 3

2653651	1598020	2492401	2245095	1620372	1960501
1690457	2067588	1976306	1976053	2645701	2067378
2380046	2265403	1865020	2235076	1450067	2205345
2508140	1976590	2120534	1756034	2458302	1972400
1478050	1583402	2400635	1650435	1832061	1870071
2235081	1576506	1890872	1760425	1976035	2063970

Pola *Corner Detection* Citra Latih 3

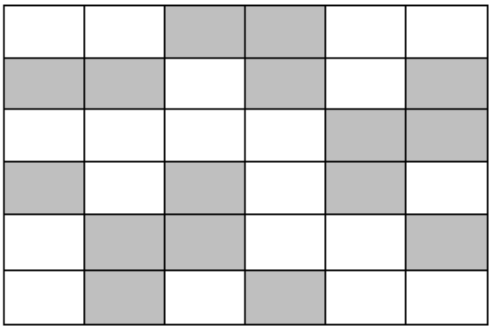


Gambar 3. 17 Pola *Corner Detection* Citra Latih 3

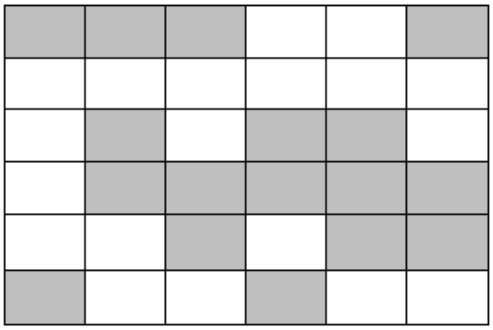
Menurut Dr. Edgar Seemann pada papernya mengenai "*Chmafer System*" bahwa Setelah mendapatkan pola *corner detection* masing-masing citra baik citra uji atau citra latih, tahap berikutnya yaitu menentukan jarak kesamaan dari pola-pola tersebut dengan cara menumpuk pola menjadi satu (pola citra latih ditumpuk dengan pola citra uji). Gambarannya sebagai berikut :

Pola deteksi jarak persamaan antara citra uji dan citra latih 1

Pola *Corner Detection* Citra Uji



Pola *Corner Detection* Citra Latih 1



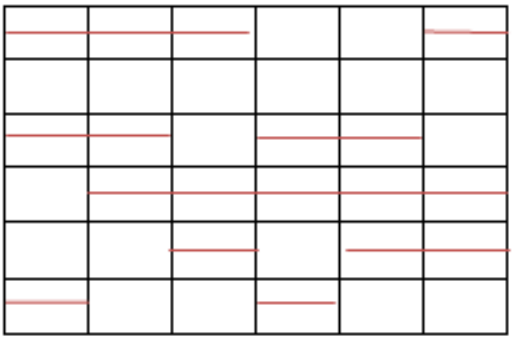
Untuk mendapatkan nilai jarak persamaan antara citra uji dengan citra latih dengan tahapan-tahapan berikut :

1. Masing-masing pola *corner* baik citra uji atau citra latih diilustrasikan sebagai berikut :

Pola *Corner Detection* Citra Uji

		0	0		
0	0		0		0
				0	0
0		0		0	
	0	0			0
	0		0		

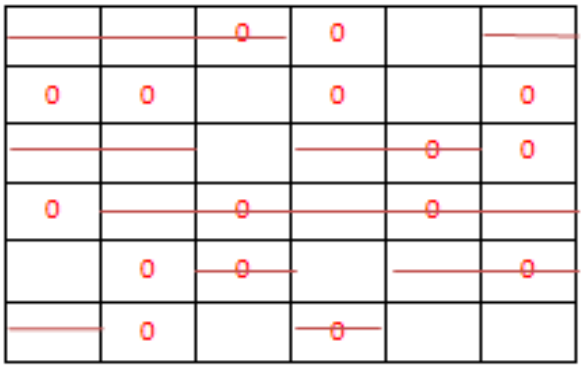
Pola *Corner Detection* Citra Latih 1



Keterangan :

- *Corner* pada citra uji di beri nilai 0
- *Corner* pada citra latih 1 di lambangkan sebagai garis

2. Tumpuk pola *corner* citra uji yang sudah di beri nilai 0 dan pola *corner* citra latih yang sudah ditandai dengan garis. Sehingga mendaptakan pola *corner* sebagai berikut :



3. Pada pola *corner* citra uji dan citra latih yang sudah di tumpuk masih terdapat *pixel* yang belum memiliki nilai yang dilalui oleh pola *corner* citra latih yang diibaratkan sebagai garis. Untuk mendapatkan nilai pada *pixel* yang belum memiliki nilai yaitu dengan cara menghitung jarak *pixel* tersebut ke *corner* citra uji (yang bernilai 0) tetangganya terdekat secara horizontal atau vertikal diambil jarak terkecil. Akan tetapi perhitungannya dilakukan dari pojok kiri atas *pixel* (0,0) ke arah kanan sampai ke *pixe*l terakhir.
4. Untuk mendapatkan nilai pada pixel (0,0) di dapatkan dari *pixel* (0,0) menuju ke *pixel* terdekat yang sudah memiliki nilai (0) yaitu kemungkinannya tetangga *pixel* yang memiliki nilai (0) ialah secara horizontal *pixel* (0,3) dan secara vertikal *pixel* (1,0). Jarak pixel (0,0) ke (0,3) adalah 3 sedangkan jarak pixel (0,0) ke (1,0) adalah 2. Sehingga jarak yang diambil jarak yang terdekat yaitu jarak antara pixel (0,0) ke pixel (1,0) adalah 2. Untuk mendapatkan semua *pixel* yang dilalui *corner* yang belum memilki nilai yaitu cara sama dengan perhitungan jarak seperti *pixel* (0,0).

   Sehingga didapatkan pola deteksi jarak persamaan corner sebagai berikut :

Pola Deteksi Jarak Persamaan *Corner* Citra Uji dengan Citra Latih 1

2	2	0	0		2
0	0		0		0
2	2		2	0	0
0	2	0	2	0	2
	0	0		2	0
2	0		0		

Gambar 3. 18 Pola *Chamfer Corner* Citra Uji dengan Citra Latih 1

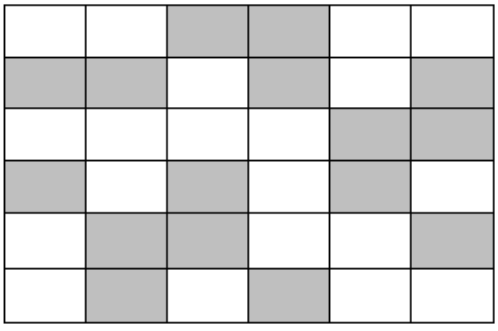
Dari gambar 3. 18 yang sudah di dapatkan jarak kesamaan nilai dari masing-masing *pixel* yang dilalui oleh *corner*. Kita bisa menghitung jumlah jarak kesamaan antara citra uji dengan citra latih 1 perhitungannya sebagai berikut :

Distance = 2+2+2+2+2+2+2+2+2+2+2 = 22

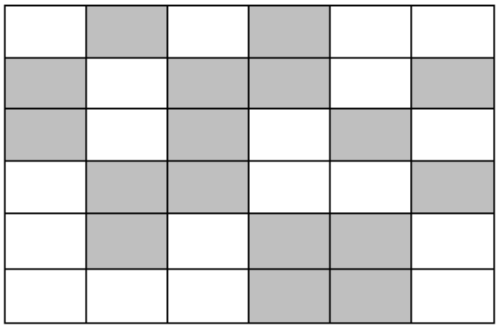
Jadi jarak kesamaan antara citra uji dan citra latih 1 adalah 2

Pola deteksi jarak persamaan antara citra uji dan citra latih 2

Pola *Corner Detection* Citra Uji

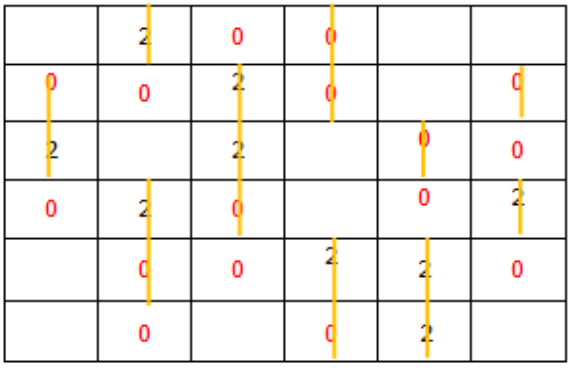


Pola *Corner Detection* Citra Latih 2



Untuk menghasilkan pola deteksi jarak persamaan *corner* citra uji dengan citra latih 2 dilakukan seperti tahapan yang dilakukan oleh pola deteksi jarak persamaan *corner* citra uji dengan citra latih 1 yang sudah dijelaskan. Sehingga didapatkan pola deteksi jarak corner sebagai berikut :

Pola Deteksi Jarak Persamaan *Corner* Citra Uji dengan Citra Latih 2



Gambar 3. 19 Pola *Chamfer Corner* Citra Uji dengan Citra Latih 2

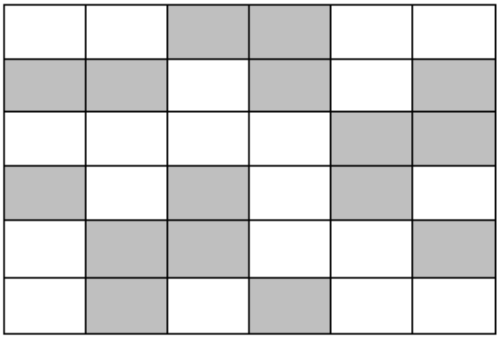
Pada pola deteksi jarak persamaan *corner* pada gambar 3.19. *corner* pada citra uji di beri nilai 0 (nol), sedangkan pada citra latih 2 yang terdeteksi *corner* ditandai dengan garis kuning. Dan pada gambar 3.19 sudah di dapatkan jarak kesamaan nilai dari masing-masing *pixel* yang dilalui oleh *corner*. Kita bisa menghitung jumlah jarak kesamaan antara citra uji dengan citra latih 2 perhitungannya sebagai berikut :

Distance = 2+2+2+2+2+2+2+2+2 = 18

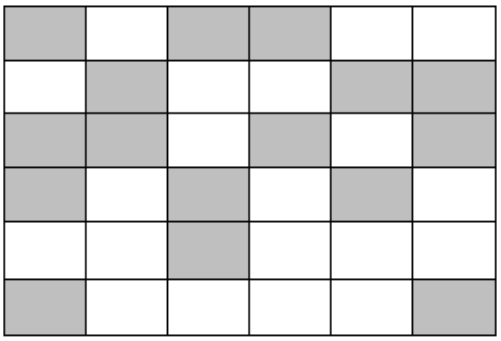
Jadi jarak kesamaan antara citra uji dan citra latih 2 adalah 18

Pola deteksi jarak persamaan antara citra uji dan citra latih 3

Pola *Corner Detection* Citra Uji



Pola *Corner Detection* Citra Latih 3



Untuk menghasilkan pola deteksi jarak persamaan *corner* citra uji dengan citra latih 3 dilakukan seperti tahapan yang dilakukan oleh pola deteksi jarak

persamaan *corner* citra uji dengan citra latih 1 yang sudah dijelaskan. Sehingga didapatkan pola deteksi jarak corner sebagai berikut :

Pola Deteksi Jarak Persamaan *Corner* Citra Uji dengan Citra Latih 3

2		0	0		
0	0		0	2	0
2	2		2	0	0
0		0		0	
	0	0			0
2	0		0		2

Gambar 3. 20 Pola *Chamfer Corner* Citra Uji dengan Citra Latih 3

Pada pola deteksi jarak persamaan *corner* pada gambar 3.20. *corner* pada citra uji di beri nilai 0 (nol), sedangkan pada citra latih 3 yang terdeteksi *corner* ditandai dengan garis biru. Dan pada gambar 3.20 sudah di dapatkan jarak kesamaan nilai dari masing-masing *pixel* yang dilalui oleh *corner*. Kita bisa menghitung jumlah jarak kesamaan antara citra uji dengan citra latih 3 perhitungannya sebagai berikut :

Distance = 2+2+2+2+2+2+2 = 14

Jadi jarak kesamaan antara citra uji dan citra latih 3 adalah 14

Tahapan mencari nilai *distance* terus dilakukan dengan cara seperti sudah dijelaskan sebelumnya, bahwasanya pola *corner* citra uji akan dibandingkan dengan semua citra latih yang ada dalam file. Adapun hasil nilai distance citra uji terhapat semua citra latih bisa dilihat pada tabel 3.2 berikut :

Tabel 3. 2 Hasil Nilai *Distance* Citra Uji dengan Citra Latih

No	Cirta Uji	Citra Latih	Nilai Distance
1	Citra uji	Citra latih 1	22
2	Citra uji	Citra latih 2	18

3	Citra uji	Citra latih 3	14
4	Citra uji	Citra latih 4	35
5	Citra uji	Citra latih 5	26
6	Citra uji	Citra latih 6	17
7	Citra uji	Citra latih 7	41
8	Citra uji	Citra latih 8	22
9	Citra uji	Citra latih 9	31
10	Citra uji	Citra latih 10	23
11	Citra uji	Citra latih 11	15
12	Citra uji	Citra latih 12	39
13	Citra uji	Citra latih 13	28
14	Citra uji	Citra latih 14	24
15	Citra uji	Citra latih 15	16
16	Citra uji	Citra latih 16	19
17	Citra uji	Citra latih 17	40
18	Citra uji	Citra latih 18	23
19	Citra uji	Citra latih 19	47
20	Citra uji	Citra latih 20	34
21	Citra uji	Citra latih 21	22
22	Citra uji	Citra latih 22	48
23	Citra uji	Citra latih 23	33
24	Citra uji	Citra latih 24	21
25	Citra uji	Citra latih 25	26
26	Citra uji	Citra latih 26	32
27	Citra uji	Citra latih 27	43
28	Citra uji	Citra latih 28	19
29	Citra uji	Citra latih 29	29
30	Citra uji	Citra latih 30	37

Dari tabel 3.2 hasil perhitungan persamaan jarak antara citra uji dan citra latih didapatkan bahwa jarak kesamaan yang terkecil adalah jarak kesamaan citra uji dengan citra latih 3 yaitu bernilai 14. Maka dapat disimpulkan bahwa citra latih yang mendekati struktur citra uji adalah citra latih 3.

Pada analisis aplikasi ini hasil perhitungan jarak tersebut untuk mendapatkan jarak kesamaan anatar dua buah gambar. Semakin rendah nilai jarak kesamaannya semakin baik hasilnya.

Algoritma untuk proses mendapatkan *feature* lalu di *matching* gambar adalah sebagai berikut :

```
Function Matching_getFeature ()

Kamus
   Matrik, height, width, threshold, i, , places : integer
   MeanX, meany, stdx, stdy, value, result: doubel
   Tmp, factor : long

Algoritma
         (*untuk mendapatkan AngularSecondMoment)
        result = 0;
        for (i = 0; i < NUMBER_OF_GRAY; i++)
            for (j = 0; j < NUMBER_OF_GRAY; j++)
                result = ( matrix[i][j], 2);
            endfor
        endfor

        (*untuk mendapatkan Contrast)
        for (i = 0; i < NUMBER_OF_GRAY; i++)
            for (j = 0; j < NUMBER_OF_GRAY; j++)
                result = (i - j, 2) *  matrix[i][j];
            endfor
        endfor

        (*untuk mendapatkan Dissimilarity)
        for (i = 0; i < NUMBER_OF_GRAY; i++)
            for (j = 0; j < NUMBER_OF_GRAY; j++)
                result = ( matrix[i][j]) *(i - j);
            endfor
        endfor

        (*untuk mendapatkan Entropy)
        for (i = 0; i < NUMBER_OF_GRAY; i++)
```

```
        for (j = 0; j < NUMBER_OF_GRAY; j++)
            if (matrix[i][j] = 0)
                result = matrix[i][j] * matrix[i][j]);
            endif
        endfor
    endfor

    (*untuk mendapatkan InverseDifferenceMoment)
    for (i = 0; i < NUMBER_OF_GRAY; i++)
        for (j = 0; j < NUMBER_OF_GRAY; j++)
            result =  matrix[i][j] / 1 +(i - j, 2);
        endfor
    endfor

    (*untuk mendapatkan Variance)
    result=0;
    mean=0;
    for(i=0; i<NUMBER_OF_GRAY; i++)
        for(j=0; j<NUMBER_OF_GRAY; j++)
            mean=matrix[i][j];
        endfor
    endfor
    mean =NUMBER_OF_GRAY*NUMBER_OF_GRAY;
    for(i=0; i<NUMBER_OF_GRAY; i++)
        for(j=0; j<NUMBER_OF_GRAY; j++)
            result =(i-mean, 2)*matrix[i][j];
        endfor
    endfor

    (*untuk mendapatkan Correlation)
    for(i=0; i<NUMBER_OF_GRAY; i++)
        for(j=0; j<NUMBER_OF_GRAY; j++)
            meanX=0;
            meanY=0;
            stdX=0;
            stdY=0;

            for(y=0; y<NUMBER_OF_GRAY; y++)
                    meanX +=matrix[y][j];
```

```
                endfor
                meanX =NUMBER_OF_GRAY;
                else
                for(x=0; x<NUMBER_OF_GRAY; x++)
                        meanY =matrix[i][x];
                endfor
                meanY =NUMBER_OF_GRAY;

                for(y=0; y<NUMBER_OF_GRAY; y++)
                    stdX =(matrix[y][j]-meanX, 2);
                endfor
                stdX=(stdX);
                else
                for(x=0; x<NUMBER_OF_GRAY; x++)
                    stdY =(matrix[i][x]-meanY, 2);
                endfor
                stdY=(stdY);

                result =(((i*j)*matrix[i][j])-(meanX*meanY))/(stdX*stdY);
            endfor
         endfor
     endfor

Function getDistance(matching feature1, matching feature2
        distance = 0;
        distance = feature1.Contrast() - feature2.Contrast(), 2);
        distance  =  feature1.Dissimilarity()  -  feature2.Dissimilarity(),
2);
        distance = feature1.Entropy() - feature2.Entropy();
        distance           =(feature1.InverseDifferenceMoment()          -
feature2.InverseDifferenceMoment(), 2);
        distance        =        (feature1.AngularSecondMoment()        -
feature2.AngularSecondMoment(), 2);

        if (places < 0)
             throw new IllegalArgumentException();
        else
         factor = (long)(10, places);
         value = value * factor;
         tmp = (value);
```

```
        return (double) tmp / factor;
    endif
endfor
```

Algoritma untuk proses membandingkan gambar uji dengan gambar latih yang ada di folder adalah sebagai berikut :

```
Function membandingkan_gambar()

Kamus
   height, width, i : integer
   ArrayList = Array_image [1...n]
   distance : double
   cImage   : image


Algoritma
  private File search(File file)
  List<Image> images = new ArrayList<Image>;
  files = file.listFile;
  for (i = 0; i < jumlah_Image; i++)
     cImage = (files[i]);
     height = cImage;
     width = cImag;

     gray = CHANNEL_GRAY;
     Matching featureTraining = HarrisFeature(gray, width, height);
     distance = 0;
     distance = Distance(featureTesting, featureTraining);

 endfor
```

3.3 Analisis Kebutuhan Sistem

Analisis kebutuhan sistem pada aplikasi yang akan dibangun terdiri dari kebutuhan fungsional dan kebutuhan non-fungsional. Adapun penjelasan lebih lanjut sebagai berikut :

3.2.3 Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional merupakan analisis yang dibutuhkan untuk menentukan spesifikasi kebutuhan yang di perlukan oleh Aplikasi. Kebutuhan non fungsional terdiri dari analisis perangkat keras, analisis perangkat lunak dan analisis pengguna (*user*). Analisis non fungsional bertujuan

agar aplikasi yang dibangun dapat digunakan sesuai dengan kebutuhan pengguna aplikasi dalam mencari inforamsi yang dibutuhkan.

3.4.1.1 Analisis Kebutuhan Perangkat Keras

Perangkat keras yang digunakan berupa sebuah laptop. Adapun spesifikasi kebutuhan perangkat keras yang dibutuhkan untuk aplikasi identifikasi arsitektur bangunan rumah tradisional Indonesia dapat dilihat pada tabel 3.3

Tabel 3. 3 Spesifikasi Kebutuhan Perangkat Keras

No	Perangkat Keras	Spesifikasi
1	Processor	Intel Core i3
2	Memori	RAM 2 GB DDR3
3	Hard Disk	320 GB
4	Monitor	Monitor 14inch

3.4.1.2 Analisis Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak untuk membangun aplikasi ini dapat dilihat pada tabel 3.4

Tabel 3. 4 Kebutuhan Perangkat Lunak

No	Perangkat Lunak	Spesifikasi
1	Sisitem Operasi	Windows 7 Profesional (64-bit)
2	Bahasa Pemograman	Java NetBean 8.0
3	Pemodelan	UML (*Unified Modeling Language*)

3.4.1.3 Analisis Pengguna Aplikasi

Analisis pengguna aplikasi ditunjukan untuk seluruh masyarakat atau seluruh orang yang ingin mengetahui arsitektur suatu bangunan, karena aplikasi ini akan mengenali jenis arsitektur bangunan pada gambar dengan

arsitektur bangunan yang sudah dikenali oleh aplikasi. Karakteristik pengguna untuk menggunakan aplikasi ini adalah:

1. Minimal dapat menjalankan aplikasi (mengoperasikan komputer)
2. Memilih gambar bangunan rumah tadisional indonesia yang akan diidentifikasi.
3. Dapat memahami konsep pengenalan gambar dengan *harris corner* dan *featur matching*

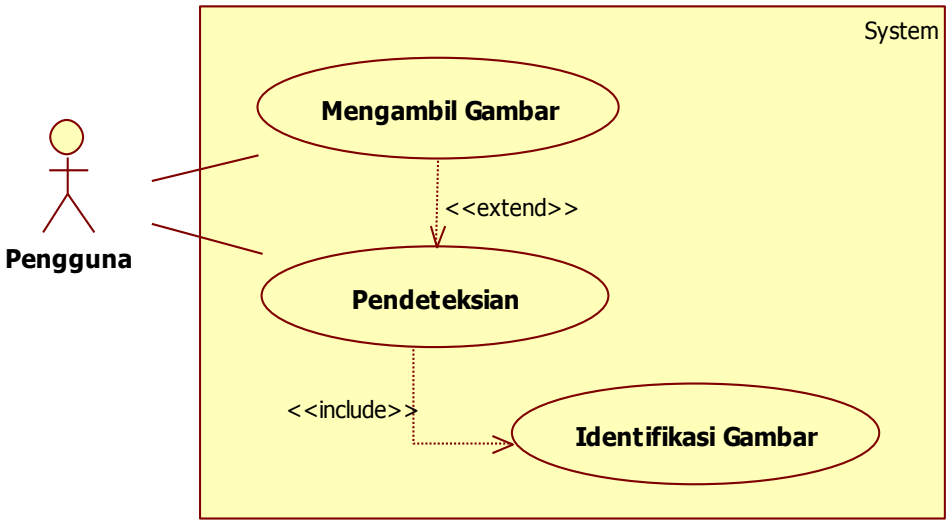
3.4.2 Analisis Kebutuhan Fungsional

Kebutuhan fungsional berhubungan dengan fitur perangkat lunak yang akan dibangun. Berikut adalah tahapan analisis kebutuhan fungsional aplikasi pengenalan arsitektur suatau bangunan pada gambar. Analisis kebutuhan fungsional menggambarkan proses kegiatan yang akan diterapkan dalam sebuah sistem dan menjelaskan kebutuhan yang diperlukan sistem agar sistem dapat berjalan dengan baik.

Analisis yang dilakukan dimodelkan dengan menggunakan UML (*Unified Modeling Language*). Tahap-tahap pemodelan dalam analisis tersebut antara lain identifikasi aktor, *Use Case diagram, skenario, sequence diagram, activity diagram,* dan *class diagram*.

3.4.2.1 *Use Case* Diagram

Use Case Diagram mendeskripsikan hubungan yang terjadi antara aktor dengan aktivitas yang terjadi pada sistem aplikasi. *Use Case* mempresentasikan operasi-operasi yang dilakukan oleh aktor dan mengidentifikasi kebutuhan fungsional. Berikut *Use Case diagram* untuk aplikasi ini dapat dilihat pada gambar 3.21 berikut :



Gambar 3. 21 *UseCase* Diagram

3.4.2.1.1 Identifikasi *Actor*

Actor adalah abstraksi dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Berikut adalah aktor yang berperan dalam menjalan aplikasi yang dibangun dapat dilihat pada tabel 3.5.

Tabel 3. 5 *Use Case Actor*

No	*Actor*	Deskripsi
A-01	Pengguna	Merupakan aktor sistem yang dibangun atau pengguna sistem

3.4.2.1.2 Identifikasi *Use Case* Diagram

Berikut tabel identifikasi *Use Case* yang terdapat pada aplikasi:

Tabel 3. 6 Identifikasi *Use Case* Diagram

No	*Use Case*	Deskripsi
UC-01	Mengambil Gambar	Fungsionalitas untuk mengambil gambar yang akan di identifikasi

UC-02	Pendeksian	Fungsionalitas untuk memproses pendeksian gambar
UC-03	Identifikasi Gambar	Fungsionalitas untuk mengidentifikasi jenis arsitektur bangunan rumah tadisional pada gambar dan menampilkan info arsitektur bangunan yang diidentifikasi

3.4.2.2 Skenario *Use Case*

Skenario *Use Case* mendeskripsikan urutan langkah-langkah dalam proses bisnis, baik yang dilakukan aktor terhadap sistem maupun yang dilakukan oleh sistem terhadap aktor.

3.4.2.2.1 Skenario *UseCase* Mengambil Gambar

Tabel 3. 7 Skenario *UseCase* Mengambil Gambar

Nama *Use Case*	Mengambil Gambar	
Aktor	Pengguna	
Tujuan	Mengambil Gambat yang akan diidentifikasi	
Kondisi Awal	Halaman Utama	
Skenario Utama		
Aksi Aktor		**Reaksi Sistem**
1. Mimilih menu Cari Gambar		
		2. Sistem menampilkan penyimpanan data gambar
3. Memilih gambar yang akan diidentifikasi		
Kondisi Akhir	Sistem menampilkan gambar yang sudah dipilih oleh pengguna	

3.4.2.2.2 Skenario *UseCase* Pendeteksian

Tabel 3. 8 Skenario *Use Case* Pendeteksian

Nama *Use Case*	Pendeteksian	
Aktor	Pengguna	
Tujuan	Melakukan Proses Pedeteksian	
Kondisi Awal	Halaman Utama	
Skenario Utama		
Aksi Aktor		**Reaksi Sistem**
1. Mimilih menu Proses		
		2. Sistem melakuan pemerosesan
Kondisi Akhir	Sistem melakukan pemerosesan	

3.4.2.2.3 Skenario *UseCase* Identifikasi Gambar

Tabel 3. 9 Skenario *UseCase* Identifikasi Gambar

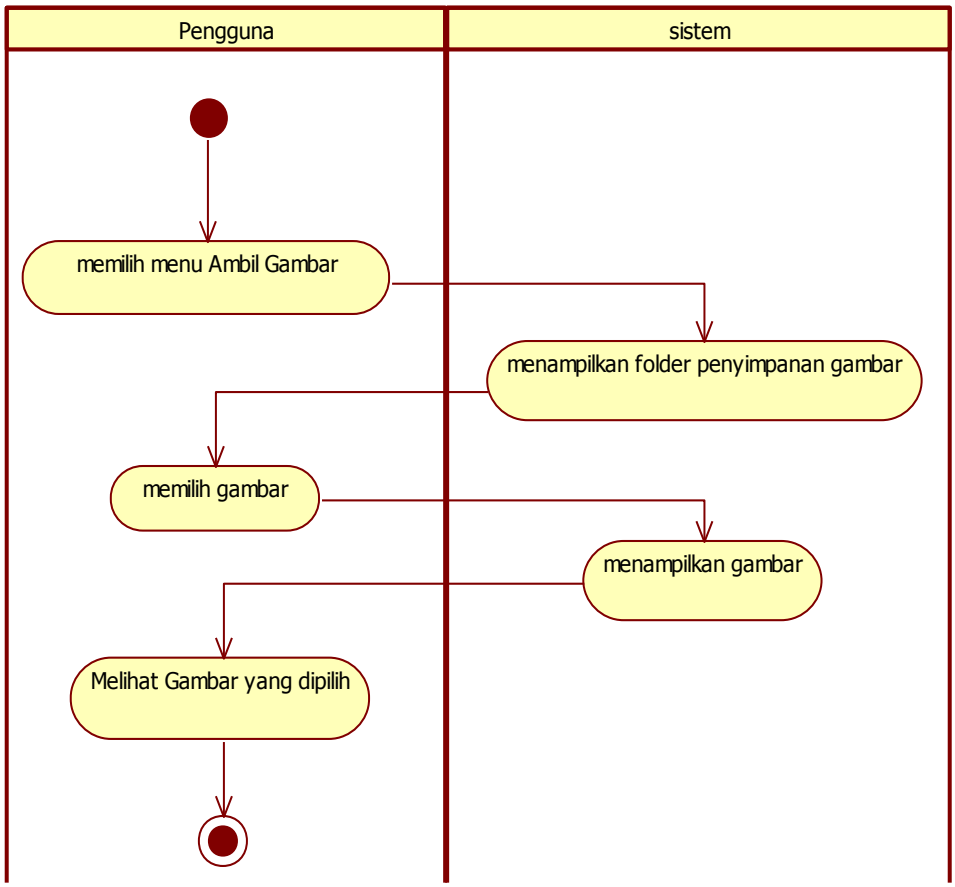
Nama *Use Case*	Identifikasi Gambar
Aktor	Pengguna
Tujuan	Melakukan tahapan-tahapan dalam pengenalan arsitektur bangunan rumah tradisional seperti analisis gambar RGB, analisis gambar *grayscale*, analisis *corner detector* dan proses *matching*. Sehingga bisa mengenali sebuah gambar
Kondisi Awal	Menu utama dengan kondisi gambar sudah dipilih
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
1. Memilih menu Proses	
	2. Sistem melakukan proses pengidentifikasian gambar (seperti analisis gambar RGB, analisis gambar *grayscale*, analisis *corner detector* dan proses *matching*.) 3. Sistem mencari detail informasi gambar

Kondisi Akhir	Sistem akan melakukan tahapan tahapan pengenalan gambar sehingga dapat mengidentifikasi arsitektur bangunan rumah tradisonal indonesia

3.4.2.3 *Activity Diagram*

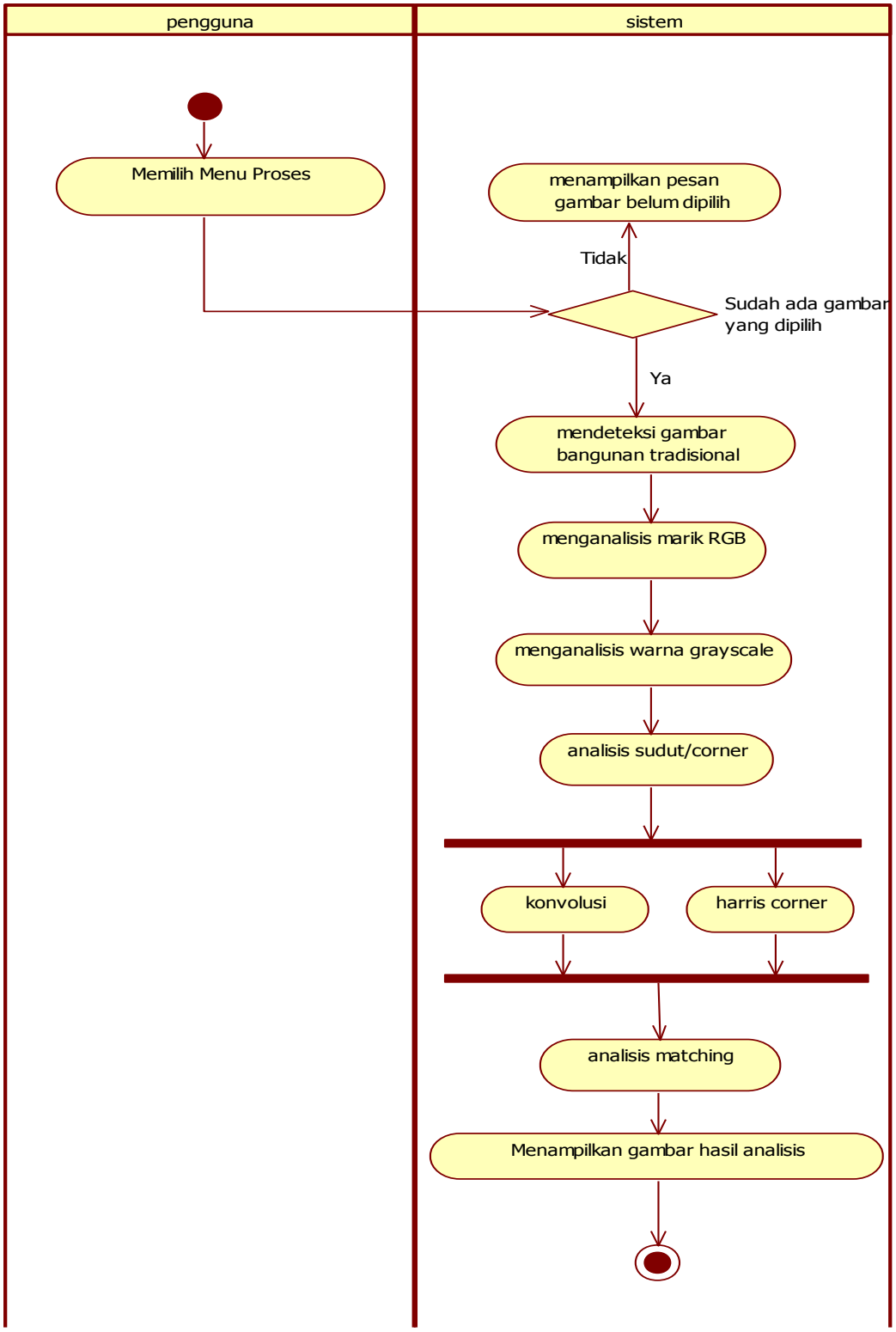
Activity Diagram merupaka s*tate* diagram khusus, dimana sebagian besar *state* adalah *action*. Menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decisio* yang mungkin terjadi. *Activity diagram* dibuat berdasarkan sebuah atau beberapa *use case* pada *use case diagram.*

3.4.2.3.1 *Activity Diagram* Mengambil Gambar



Gambar 3. 22 *Activity* Diagram Ambil Gambar

3.4.2.3.2 *Activity* Diagram Identifikasi Gambar

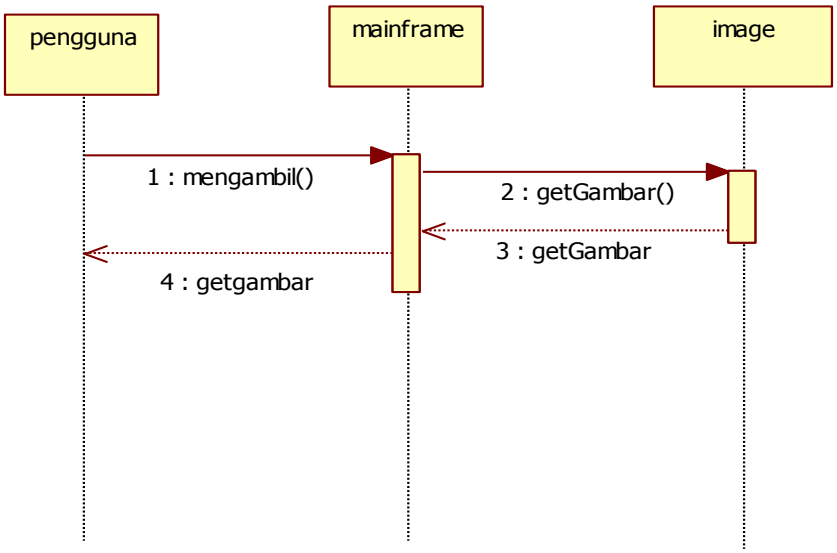


Gambar 3. 23 *Activity* Diagram Identifikasi Gambar

3.4.2.4 *Sequence Diagram*

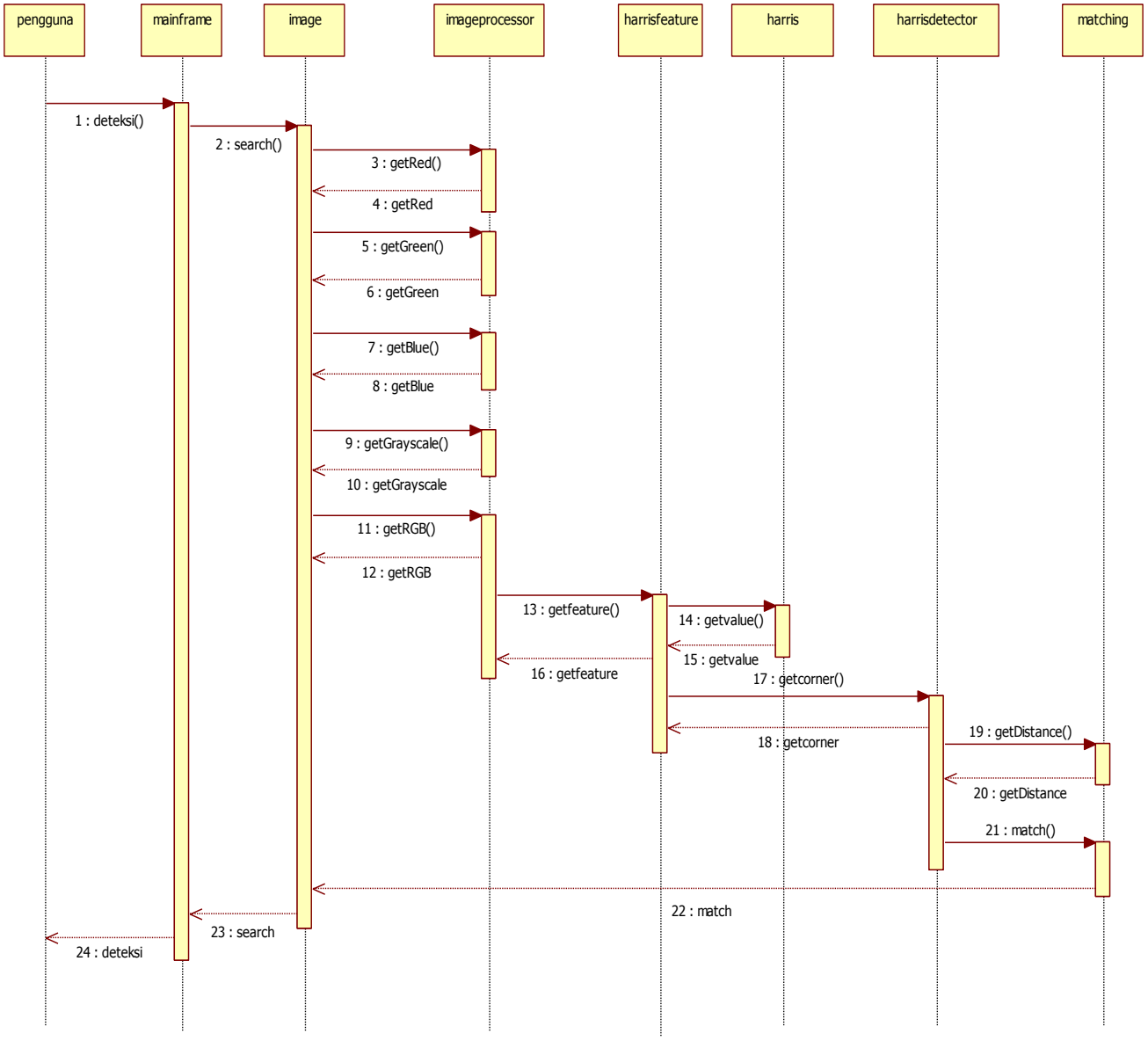
Sequence diagram menggambarkan perilaku pada sebuah skenario. Diagram ini menunjukkan sejumlah contoh obyek dan *message* (pesan) yang diletakkan diantara obyek-obyek ini di dalam *Use Case*. Komponen utama *sequence diagram* terdiri atas obyek yang dituliskan dengan kotak segiempat. *Message* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukkan dengan progres vertikal.

3.4.2.4.1 *Sequence Diagram* Mengambil Gambar



Gambar 3. 24 *Sequence* Diagram Mengambil Gambar

3.4.2.4.2 *Sequence Diagram* Identifikasi Gambar

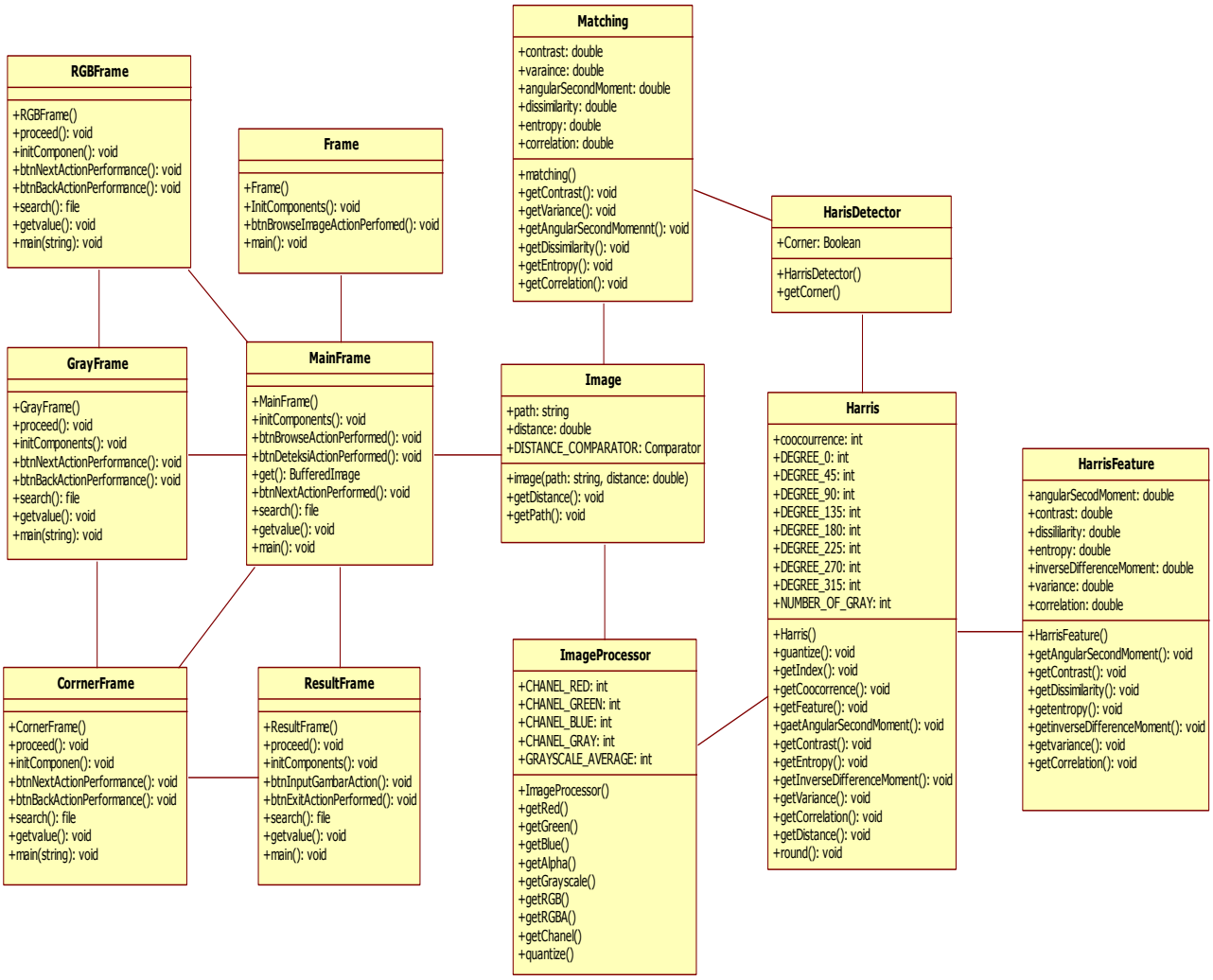


Gambar 3. 25 *Sequence* Diagram Identifikasi Gambar

3.4.2.5 *Class Diagram*

Class diagram merupakan inti dari pemograman berbasis objek karena diagram ini memberikan petaan terhadap kelas-kelas yang digunakan oleh suatu aplikasi. *Class diagram* menggambarkan struktur dan hubungan antar objek-objek yang ada pada sistem. Struktur itu meliputi atribut-atribut dan

metode-metode yang ada pada masing-masing *class*. Adapun aplikasi identifikasi arsitektur bangunan ini memiliki kelas diagram sebagai berikut:



Gambar 3. 26 *Class* Diagram

Tabel 3. 10 Deskripsi *Class Diagram*

Class	Jenis Class	Deskripsi
Frame	*Boundary*	Merupakan class yang digunakan untuk menampilkan MainFrame
MainFrame	*View*	Menampilkan *class* yang digunakan untuk menampilkan menu

Image	*Entity*	Merupakan *class* yang berisi informasi dari gambar
Image Processor	*Control*	Merupakan *class* yang berisi method-method untuk pengenalan gambar dan perubahan warna gambar
Harris	*Control*	Merupakan *class* yang berisi informasi *feature detection* didapatkan dari *class harris*
Harris Feature	*Control*	Merupakan *class* yang berisi method-method untuk mendapatkan nilai yang terdeteksi sebagai *feature*
Harris Detector	*Control*	Merupakan *class* yang berisi method-method untuk mendapatkan *corner* atau deteksi *corner*
Matching	*Control*	Merupakan *class* yang digunakan untuk mecari persamaan (*match*) antara gambar gambar
RGBFrame	*View*	Menampilkan *class* yang digunakan untuk menampilkan gambar hasil konvolusi dari gambar RGB menjadi masing-masing gambar *Red, Green dan Blue.*
GrayFrame	*View*	Menampilkan *class* yang digunakan untuk menampilkan gambar *Grayscale*
CornerFrame	*View*	Menampilkan *class* yang digunakan untuk menampilkan gambar yang telah dideteksi *Corner*
ResultFrame	*View*	Menampilkan *class* yang digunakan untuk menampilkan hasil pengidentifikasian gambar bangunan rumah tradisional Indonesia

3.5 Perancangan Sistem

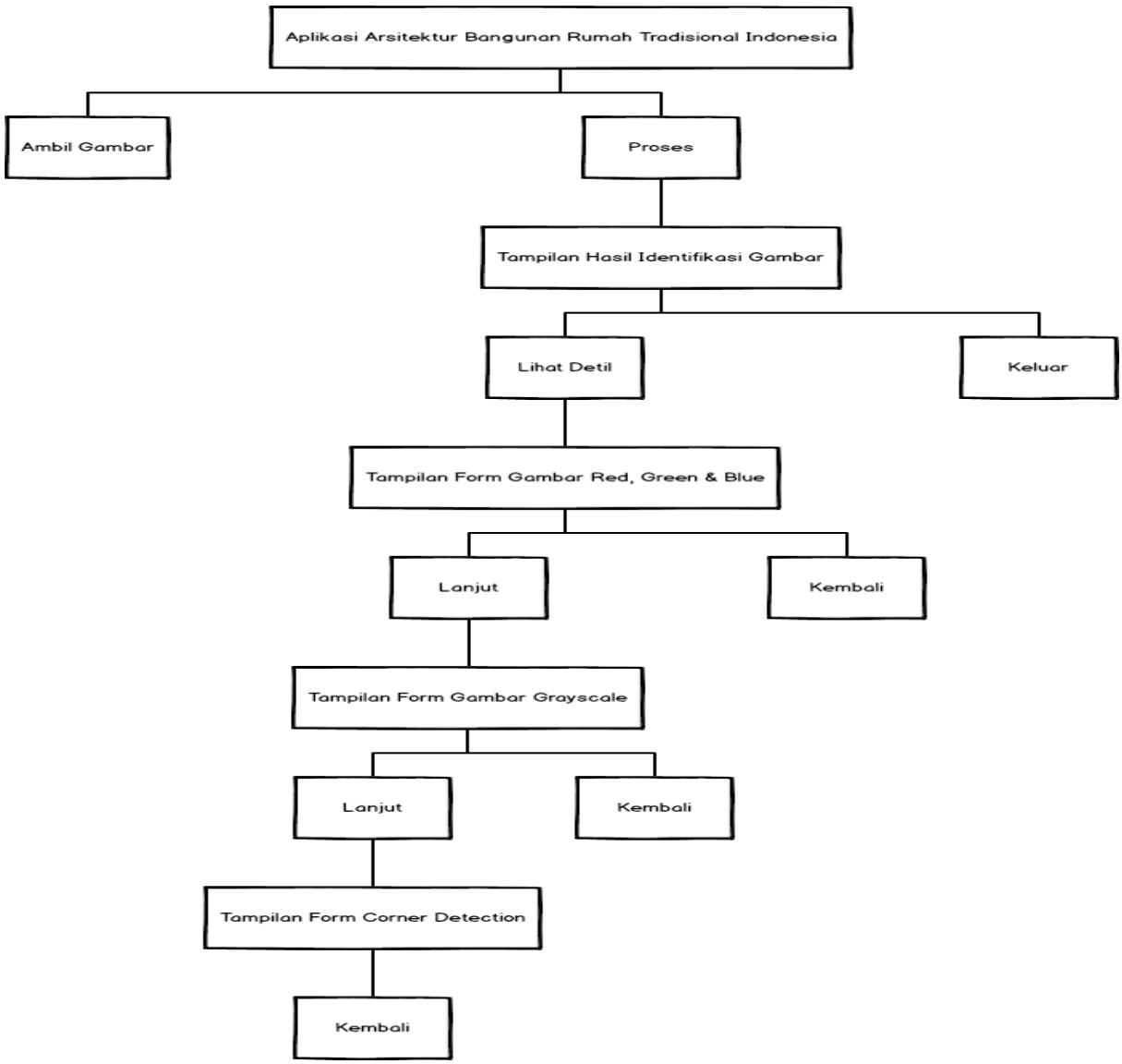
Perancangan sistem merupaka suatu proses yang menggambarkan bagaimana suatu sistem dibangun untuk memenuhi kebutuhan pada fase analisis. Tahap perancangan arsitektur terdiri dar perancangan arsitektur.

3.5.1 Perancangan Arsitektur

Perancangan arsitektur terdiri dari perancangan struktur menu, perancangan antarmuka, perancangan pesan dan perancangan jaringan sematik.

3.5.1.1 Perancangan Struktur Menu

Perancangan menu dilakukan untuk mempermudah interaksi antara sistem dengan pengguna. dapat digambarkan sebagai berikut :

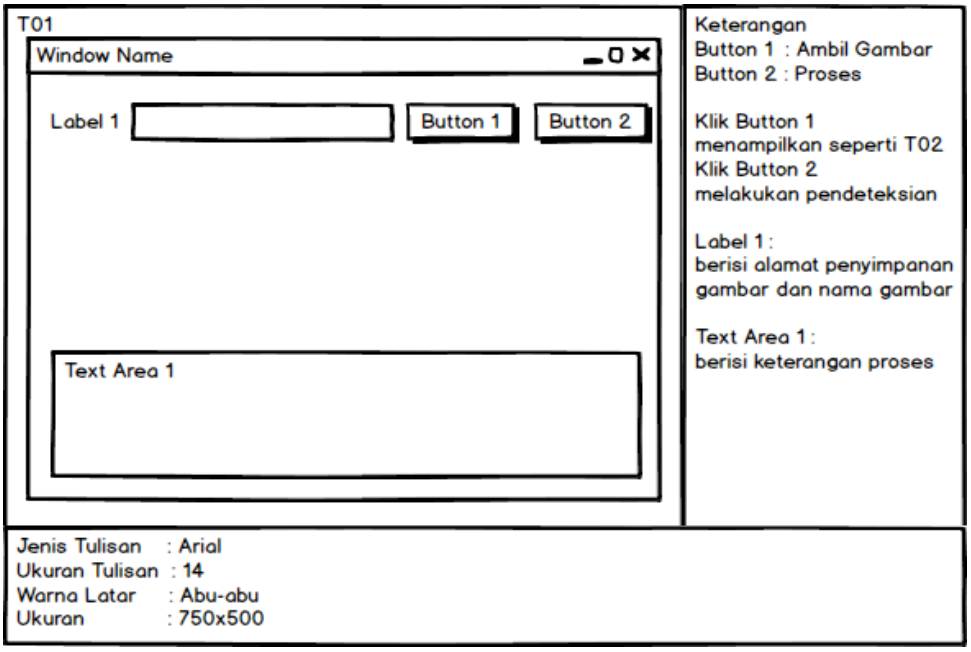


Gambar 3. 27 Perancangan Struktur Menu

3.5.1.2 Perancangan Antarmuka Sistem

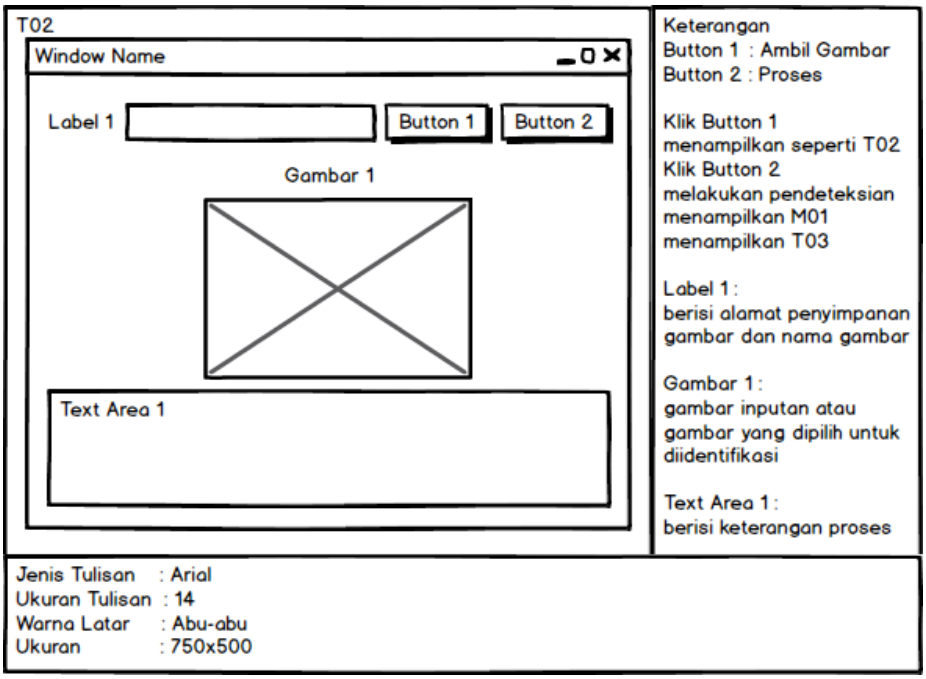
Tahap Perancangan antarmuka bertujuan untuk memberikan gambaran tentang aplikasi yang akan dibangun, sehingga akan mempermudah dalam mengimplementasikan aplikasi serta akan memudahkan pembuatan aplikasi. Adapun perancangan antarmuka perangkat lunak identifikasi arsitektur bangunan pada gambar ini adalah sebagai berikut :

3.5.1.2.1 Perancangan Antarmuka Awal



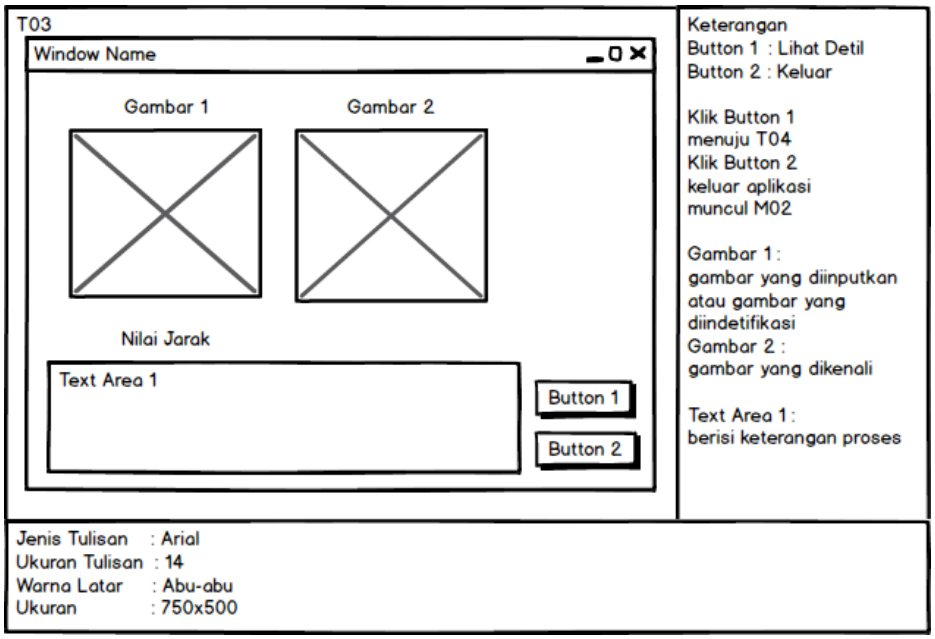
Gambar 3. 28 Perancangan Antarmuka Awal

3.5.1.2.2 Perancangan Antarmuka Setelah Memilih Gambar



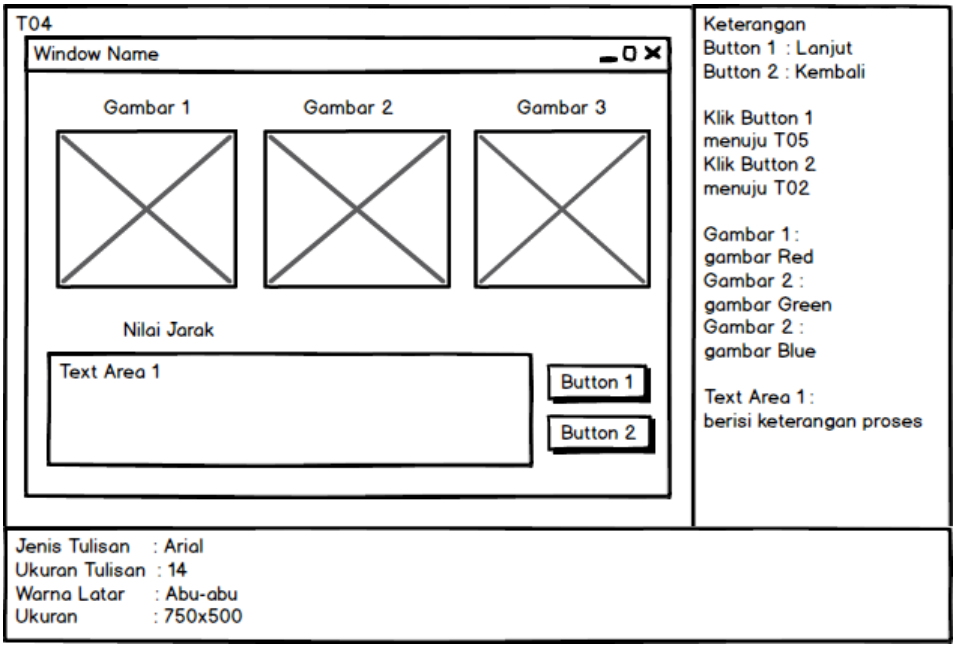
Gambar 3. 29 Perancangan Antarmuka Setelah Memilih Gambar

3.5.1.2.3 Perancangan Antarmuka Hasil Identifikasi Gambar



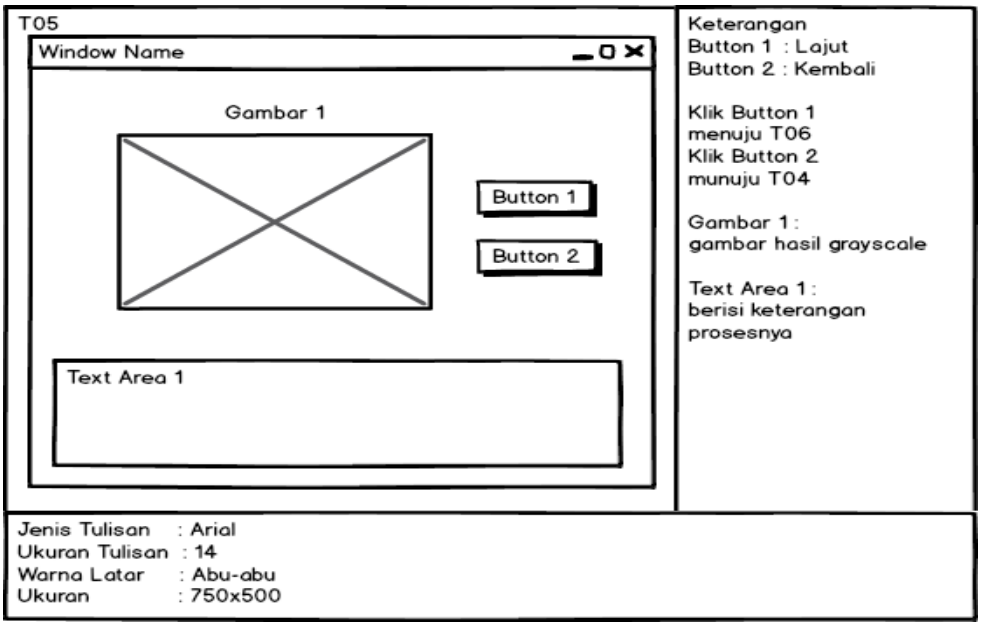
Gambar 3. 30 Perancangan Antarmuka Hasil Identifikasi Gambar

3.5.1.2.4 Perancangan Antarmuka Tampilan Gambar Red, Green dan Blue



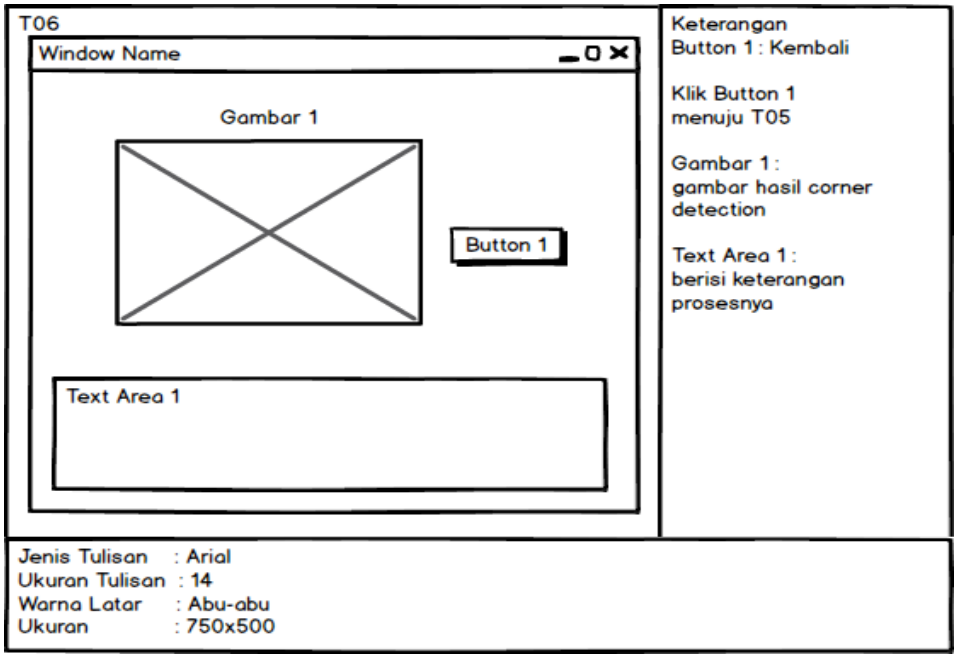
Gambar 3. 31 Perancangan Antarmuka Tampilan Gambar *Red, Green* dan *Blue*

3.5.1.2.5 Perancangan Antarmuka Tampilan Gambar Hasil *Grayscale*



Gambar 3. 32 Perancangan Antarmuka Tampilan Gambar Hasil *Grayscale*

3.5.1.2.6 Perancangan Antarmuka Tampilan Gambar Hasil *Corner Detection*



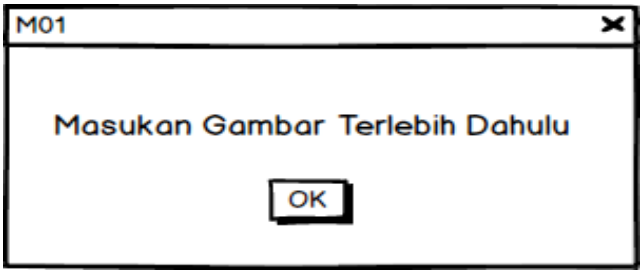
Gambar 3. 33 Perancangan Antarmuka Tampilan Gambar *Hasil Corner Detection*

3.5.1.3 Perancangan Pesan

Pesan merupakan tampilan dari suatu perangkat lunak yang berfungsi untuk menyampaikan notifikasi dan informasi kepada pengguna agar perangkat lunak lebih interaktif. Perancangan pesan pada aplikasi pengidentifikasian arsitektur bangunan tradisional indonesia terdapat 3 perancangan pesan

3.5.1.3.1 Perancangan Pesan Jika Belum Memilih Gambar

Pesan ini muncul jika pengguna menekan tombol proses sebelum memilih gambar. Perancangan pesannya sebagai berikut :



Gambar 3. 34 Perancangan Pesan Jika Belum Memilih Gambar

3.5.1.3.2 Perancangan Pesan Ketika Ingin Keluar Aplikasi

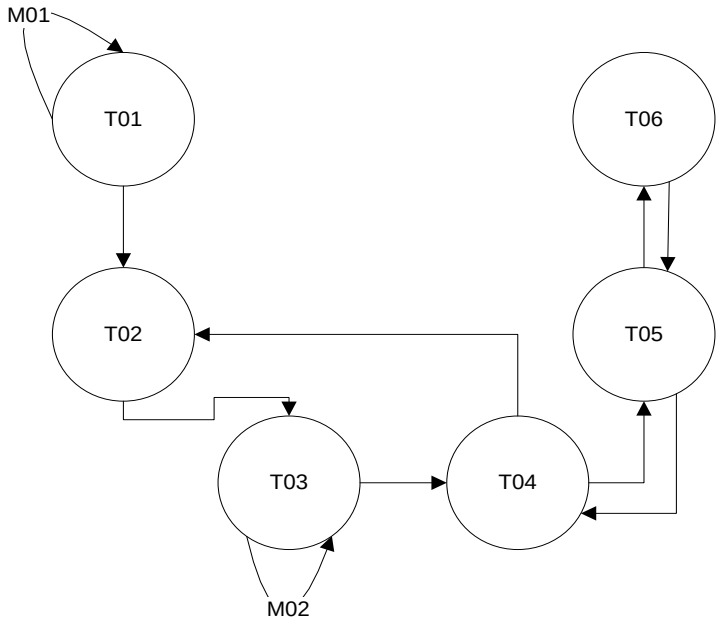
Pesan ini muncul apabila ketika tekan tombol keluar dari aplikasi



Gambar 3. 35 Perancangan Pesan Ketika Ingin Keluar Aplikasi

3.5.1.4 Perancangan Jaringan Semantik

Jaringan semantik adalah diagram yang menggambarkan aliran-aliran menu dan pesan dalam sebuah program. Jaringan sematik dari aplikasi identifikasi arsitekur bangunan yang dibangun yaitu sebagai berikut :



Gambar 3. 36 Jaringan Semantik Aplikasi Identifikasi Arsitektur Bangunan Rumah Tradisional Indonesia