BAB IV
IMPLEMENTASI

Pada bab implementasi akan dibahas mengenai implementasi sistem yang dibuat baik itu perangkat keras maupun perangkat lunak, batasan-batasan implementasi, implementasi program dari proses binerisasi, proses erosi, proses skeletonisasi, sampai dengan parameter HMM, implementasi *database* dan implementasi *interface.*

4.1 Implementasi Sistem

Perangkat lunak pengenalan sidik jari dikembangkan dalam lingkungan implementasi yang terdiri dari perangkat keras dan perangkat lunak.

4.1.1 Implementasi Perangkat Keras

Implementasi perangkat keras yang dipakai dalam proses pembuatan sisitem dijelaskan pada tabel berikut ini :

Tabel 4.1 Implementasi perangkat keras komputer

Notebook Toshiba Satellite L635	
Nama *Hardware*	Spesifikasi
Processor	Intel (R) Core (TM) i5 M460 2.53 GHz
Memory (RAM)	2 GB
Harddisk	Seagate Momentus 5400.6 SATA 3Gb/s 500 GB
Motherboard	Toshiba Notebook Intel Motherboard
Graphic Card	ATI Mobility Radeon HD 5470
Monitor	13.3" Widescreen LED Backlit Display
Webcam	Toshiba Web Camera 1,3 MP

4.1.2 Implementasi Perangkat Lunak

Implementasi perangkat lunak yang dipakai dalam proses pembuatan sistem dijelaskan pada tabel berikut ini :

Tabel 4.2 Implementasi perangkat lunak komputer

Notebook Toshiba Satellite L635	
Nama *software*	Spesifikasi
Sistem operasi	Microsoft Windows 7 Ultimate 32-bit
Versi *DirectX*	DirectX 11

Bahasa pemrograman	C#
Integrated Development Environment	Microsoft Visual Studio 2008
Database Management System	MySQL XAMPP 1.7.7

4.2 Batasan – Batasan Implementasi

Beberapa batasan dalam mengimplementasikan perangkat lunak pengenalan sidik jari adalah sebagai berikut :

1. Perangkat lunak pengenalan sidik jari dirancang dan dijalankan dengan menggunakan *Desktop Application*.
2. *File* yang digunakan memiliki format **.tiff*.
3. *Database Management System* yang digunakan adalah MySQL.
4. Setiap *file* mempunyai *filename* xxx_y_z dimana x adalah *person ID*, y adalah *finger ID*, dan z adalah *number of scan*
5. Ukuran *width* dan *height* citra adalah 504 x 408 piksel.

4.3 Implementasi Program

Terdapat beberapa proses dalam program baik dalam program *training* maupun program *testing*. Adapun proses-prosesnya adalah sebagai berikut :

4.3.1 Implementasi Proses Binerisasi

Proses binerisasi ini bertujuan untuk mengubah citra awal yang berwarna grayscale menjadi citra berwarna hitam dan putih. Proses implementasi proses binerisasi seperti yang ditunjukkan pada *Sourcecode* 4.1 di bawah ini :

Sourcecode 4.1 Implementasi proses binerisasi

```
1   red = green = blue = new int[bmp.Width, bmp.Height];
2   for (int i = 0; i < bmp.Width; i++)
3   {
4   for (int j = 0; j < bmp.Height; j++)
5   {
6   red[i, j] = bmp.GetPixel(i, j).R;
7   green[i, j] = bmp.GetPixel(i, j).G;
8   blue[i, j] = bmp.GetPixel(i, j).B;
9   }
10  }
11
12  for (int i = 0; i < bmp.Width; i++)
13  for (int j = 0; j < bmp.Height; j++)
```

```
14  {
15  p = red[i, j];
16  histogram[p]++;
17  }
18  box1_img_tes.Image = bmp;
19  bmp = new Bitmap(box1_img_tes.Image);
20
21  for (k = 0; k < 256; k++)
22  tmean += k * histogram[k] / (bmp.Width * bmp.Height);
23  for (k = 0; k < 256; k++)
24  {
25  zerothcm += histogram[k] / (bmp.Width * bmp.Height);
26  firstcm += k * histogram[k] / (bmp.Width * bmp.Height);
27  variance = (tmean * zerothcm - firstcm);
28  variance *= variance;
29  variance /= zerothcm * (1 - zerothcm);
30  if (maxvariance < variance)
31  {
32  maxvariance = variance;
33  t = k;
34  }
35  }
36
37  for (int i = 0; i < bmp.Width; i++)
38  for (int j = 0; j < bmp.Height; j++)
39  {
40  p = red[i, j];
41  if (p >= t) bmp.SetPixel(i, j, Color.FromArgb(255, 255,
42  255));
43  else bmp.SetPixel(i, j, Color.FromArgb(0, 0, 0));
44  }
```

Penjelasan dari *Sourcecode* 4.1 proses binerisasi sebagai berikut ini :

Baris 1-11 adalah proses memasukkan ukuran citra kedalam array kemudian ditelusuri.

Baris 13-20 adalah proses pembuatan histogram.

Baris 22-39 adalah metode otsu dengan mencari nilai variance tertinggi dan diambil nilai k untuk dijadikan threshold.

Baris 41-49 adalah proses merubah gambar menjadi bernilai 0 dan 255.

4.3.2 Implementasi Proses Erosi

Proses Erosi bertujuan untuk menghilangkan noise-noise pada citra dan untuk memperbaiki struktur citra agar terlihat lebih baik. Implementasi proses erosi ditunjukkan pada *Sourcecode* 4.2 berikut ini :

Sourcecode 4.2 Implementasi proses erosi

```
1   private Bitmap erosi(Bitmap bmp)
2   {
3   red = green = blue = new int[bmp.Width, bmp.Height];
```

```
4   for (int i = 1; i < bmp.Width - 1; i++)
5   {
6   for (int j = 1; j < bmp.Height - 1; j++)
7   {
8   red[i, j] = bmp.GetPixel(i, j).R;
9   green[i, j] = bmp.GetPixel(i, j).G;
10  blue[i, j] = bmp.GetPixel(i, j).B;
11  }
12  }
13
14  for (int c = 0; c < 1; c++)
15  {
16  bmp1 = new Bitmap(bmp);
17  for (int i = 1; i < bmp.Width - 1; i++)
18  for (int j = 1; j < bmp.Height - 1; j++)
19  {
20  if (red[i - 1, j - 1] == 0 && red[i - 1, j] == 0 && red[i -
21  1, j + 1] == 0 && red[i, j - 1] == 0 && red[i, j] == 0 &&
22  red[i, j + 1] == 0 && red[i + 1, j - 1] == 0 && red[i + 1,
23  j] == 0 && red[i + 1, j + 1] == 0)
24  bmp1.SetPixel(i, j, Color.FromArgb(0, 0, 0));
25  else bmp1.SetPixel(i, j, Color.FromArgb(255, 255, 255));
26  }
27  bmp = new Bitmap(bmp1);
28  box1_img_tes.Image = (Image)bmp;
29  }
30  return bmp;
31  }
```

Penjelasan dari *Sourcecode* 4.2 proses erosi sebagai berikut ini :

Baris 3-13 adalah proses mengambil dan memasukkan nilai piksel citra kedalam array.

Baris 15-36 adalah inti dari proses erosi yaitu proses pengecekan antara nilai piksel 8-ketetanggaan dengan nilai SE.

4.3.3 Implementasi Proses Skeletonisasi

Proses skeletonisasi adalah proses mengubah citra menjadi setebal 1 piksel. Proses ini bertujuan untuk memudahkan dalam mencari ekstraksi fitur dan nama metode yang digunakan dalam skeletonisasi adalah metode Zhang-Suen. Adapun proses implmentasi dari proses skeletonisasi ditunjukkan pada *Sourcecode* 4.3 berikut ini :

Sourcecode 4.3 Implementasi proses skeletonisasi

```
1   private void skeletonisasi_tes_Click(object sender,
2   EventArgs e)
3   {
4   int count = 0;
5   bmp1 = new Bitmap(bmp);
6   do
```

```
7   {
8   count = 0;
9   for (int j = 1; j < bmp.Height - 1; j++)
10  for (int i = 1; i < bmp.Width - 1; i++)
11  {
12  if (bmp.GetPixel(i, j).R == 0)
13  {
14  if (cekbp(i, j) == true && ap(i, j) == true && cek1(i, j) ==
15  true && cek2(i, j) == true)
16  {
17  bmp1.SetPixel(i, j, Color.White);
18  count++;
19  }
20  if (cekbp(i, j) == true && ap(i, j) == true && cek3(i, j) ==
21  true && cek4(i, j) == true)
22  {
23  bmp1.SetPixel(i, j, Color.White);
24  count++;
25  }
26  }
27  }
28  bmp = new Bitmap(bmp1);
29  }
30  while (count != 0);
31  box1_img_tes.Image = (Image)bmp;
32  }
33  private bool cekbp(int x, int y)
34  {
35  int jumlahbp = 0;
36  for (int i = x - 1; i <= x + 1; i++)
37  for (int j = y - 1; j <= y + 1; j++)
38  {
39  if (i != x && j != y)
40  if (bmp.GetPixel(i, j).R != 255)
41  jumlahbp++;
42  }
43  if (jumlahbp >= 2 && jumlahbp <= 6)
44  return true;
45  else return false;
46  }
47
48  private bool ap(int x, int y)
49  {
50  int perubahan = 0;
51  if (bmp.GetPixel(x, y - 1).R == 255 && bmp.GetPixel(x + 1, y
52  - 1).R == 0)
53  perubahan++;
54  if (bmp.GetPixel(x + 1, y - 1).R == 255 && bmp.GetPixel(x +
55  1, y).R == 0)
56  perubahan++;
57  if (bmp.GetPixel(x + 1, y).R == 255 && bmp.GetPixel(x + 1, y
58  + 1).R == 0)
59  perubahan++;
60  if (bmp.GetPixel(x + 1, y + 1).R == 255 && bmp.GetPixel(x, y
61  + 1).R == 0)
62  perubahan++;
63  if (bmp.GetPixel(x, y + 1).R == 255 && bmp.GetPixel(x - 1, y
```

```
64  + 1).R == 0)
65  perubahan++;
66  if (bmp.GetPixel(x - 1, y + 1).R == 255 && bmp.GetPixel(x -
67  1, y).R == 0)
68  perubahan++;
69  if (bmp.GetPixel(x - 1, y).R == 255 && bmp.GetPixel(x, y -
70  1).R == 0)
71  perubahan++;
72  if (bmp.GetPixel(x - 1, y - 1).R == 255 && bmp.GetPixel(x -
73  1, y).R == 0)
74  perubahan++;
75  if (perubahan == 1)
76  return true;
77  else return false;
78  }
79
80  private bool cek1(int x, int y)
81  {
82  if (bmp.GetPixel(x, y - 1).R == 0 && bmp.GetPixel(x + 1,
83  y).R == 0 && bmp.GetPixel(x, y + 1).R == 0 ||
84  bmp.GetPixel(x, y - 1).R == 255 && bmp.GetPixel(x + 1, y).R
85  == 255 && bmp.GetPixel(x, y + 1).R == 255)
86  return false;
87  else return true;
88  }
89
90  private bool cek2(int x, int y)
91  {
92  if (bmp.GetPixel(x + 1, y).R == 0 && bmp.GetPixel(x, y +
93  1).R == 0 && bmp.GetPixel(x - 1, y).R == 0 ||
94  bmp.GetPixel(x + 1, y).R == 255 && bmp.GetPixel(x, y + 1).R
95  == 255 && bmp.GetPixel(x - 1, y).R == 255)
96  return false;
97  else return true;
98  }
99
100 private bool cek3(int x, int y)
101 {
102 if (bmp.GetPixel(x, y - 1).R == 0 && bmp.GetPixel(x + 1,
103 y).R == 0 && bmp.GetPixel(x - 1, y).R == 0 ||
104 bmp.GetPixel(x, y - 1).R == 255 && bmp.GetPixel(x + 1, y).R
105 == 255 && bmp.GetPixel(x - 1, y).R == 255)
106 return false;
107 else return true;
108 }
109
110 private bool cek4(int x, int y)
111 {
112 if (bmp.GetPixel(x, y - 1).R == 0 && bmp.GetPixel(x, y +
113 1).R == 0 && bmp.GetPixel(x - 1, y).R == 0 ||
114 bmp.GetPixel(x, y - 1).R == 255 && bmp.GetPixel(x, y + 1).R
115 == 255 && bmp.GetPixel(x - 1, y).R == 255)
116 return false;
117 else return true;
118 }
```

Penjelasan *Sourcecode* 4.3 proses skeletonisasi sebagai berikut :

Baris 1-33 adalah proses utama dalam skeletonisasi.

Baris 35-48 adalah proses pengecekan jumlah 255 lebih dari sama dengan 2 atau kurang dari sama dengan 6 jika benar maka true jika salah false

Baris 50-80 adalah pengecekan perpindahan dari 255 ke 0 jika lebih dari satu maka true jika tidak maka false

Baris 82-91 adalah pengecekan nilai P2*P4*P6=0

Baris 93-103 adalah pengecekan nilai P4*P6*P8=0

Baris 105-114 adalah pengecekan nilai P2*P4*P8=0

Baris 116-126 adalah pengeceken nilai P2*P6*P8=0

4.3.4 Implementasi Proses Ekstraksi Fitur

Proses ekstraksi fitur bertujuan untuk mengambil berbagai macam fitur yang ada pada semua citra sidik jari. Dalam proses ini ekstraksi fitur menggunakan percabangan (*bifurcation*) sejumlah 24 kemungkinan *bifurcation*. Adapun proses implementasi untuk proses ekstraksi fitur ditunjukkan pada *Sourcecode* 4.4 berikut ini :

Sourcecode 4.4 Implementasi proses ekstraksi fitur

```
1   private void EF_testing_Click(object sender, EventArgs e)
2   {
3   red = green = blue = new int[bmp.Width, bmp.Height];
4   for (int i = 1; i < bmp.Height - 1; i++)
5   {
6   for (int j = 1; j < bmp.Width - 1; j++)
7   {
8   red[j, i] = bmp.GetPixel(j, i).R;
9   green[j, i] = bmp.GetPixel(j, i).G;
10  blue[j, i] = bmp.GetPixel(j, i).B;
11  }
12  }
13
14  bmp1 = new Bitmap(bmp);
15  String pola = "";
16  int count = 0;
17  for (int i = 1; i < bmp.Height - 1; i++)
18  for (int j = 1; j < bmp.Width - 1; j++)
19  {
20  if (cek_bif(0, 255, 0, 255, 0, 255, 255, 0, 255, j, i) == 9)
21  {
22  pola += "0,";
23  rectangle(bmp1, j, i, 3, 3);
24  count++;
25  }
```

```
26  else if (cek_bif(255, 0, 255, 255, 0, 255, 0, 255, 0, j, i)
27  == 9)
28  {
29  pola += "1,";
30  rectangle(bmp1, j, i, 3, 3);
31  count++;
32  }
33  else if (cek_bif(0, 255, 255, 255, 0, 0, 0, 255, 255, j, i)
34  == 9)
35  {
36  pola += "2,";
37  rectangle(bmp1, j, i, 3, 3);
38  count++;
39  }
40  else if (cek_bif(255, 255, 0, 0, 0, 255, 255, 255, 0, j, i)
41  == 9)
42  {
43  pola += "3,";
44  rectangle(bmp1, j, i, 3, 3);
45  count++;
46  }
47  else if (cek_bif(255, 0, 255, 0, 255, 0, 255, 255, 255, j,
48  i) == 9)
49  {
50  pola += "4,";
51  rectangle(bmp1, j, i, 3, 3);
52  count++;
53  }
54  else if (cek_bif(255, 255, 255, 0, 255, 0, 255, 0, 255, j,
55  i) == 9)
56  {
57  pola += "5,";
58  rectangle(bmp1, j, i, 3, 3);
59  count++;
60  }
61  else if (cek_bif(255, 0, 255, 0, 255, 255, 255, 0, 255, j,
62  i) == 9)
63  {
64  pola += "6,";
65  rectangle(bmp1, j, i, 3, 3);
66  count++;
67  }
68  else if (cek_bif(255, 0, 255, 255, 255, 0, 255, 0, 255, j,
69  i) == 9)
70  {
71  pola += "7,";
72  rectangle(bmp1, j, i, 3, 3);
73  count++;
74  }
75  else if (cek_bif(0, 255, 0, 255, 0, 255, 0, 255, 255, j, i)
76  == 9)
77  {
78  pola += "8,";
79  rectangle(bmp1, j, i, 3, 3);
80  count++;
81  }
82  else if (cek_bif(0, 255, 255, 255, 0, 255, 0, 255, 0, j, i)
```

```
83  == 9)
84  {
85  pola += "9,";
86  rectangle(bmp1, j, i, 3, 3);
87  count++;
88  }
89  else if (cek_bif(0, 255, 0, 255, 0, 255, 255, 255, 0, j, i)
90  == 9)
91  {
92  pola += "10,";
93  rectangle(bmp1, j, i, 3, 3);
94  count++;
95  }
96  else if (cek_bif(255, 255, 0, 255, 0, 255, 0, 255, 0, j, i)
97  == 9)
98  {
99  pola += "11,";
100 rectangle(bmp1, j, i, 3, 3);
101 count++;
102 }
103 else if (cek_bif(255, 0, 255, 0, 255, 0, 0, 255, 255, j, i)
104 == 9)
105 {
106 pola += "12,";
107 rectangle(bmp1, j, i, 3, 3);
108 count++;
109 }
110 else if (cek_bif(255, 0, 255, 0, 255, 0, 255, 255, 0, j, i)
111 == 9)
112 {
113 pola += "13,";
114 rectangle(bmp1, j, i, 3, 3);
115 count++;
116 }
117 else if (cek_bif(0, 225, 255, 0, 255, 0, 255, 0, 255, j, i)
118 == 9)
119 {
120 pola += "14,";
121 rectangle(bmp1, j, i, 3, 3);
122 count++;
123 }
124 else if (cek_bif(255, 255, 0, 0, 255, 0, 255, 0, 255, j, i)
125 == 9)
126 {
127 pola += "15,";
128 rectangle(bmp1, j, i, 3, 3);
129 count++;
130 }
131 else if (cek_bif(255, 255, 0, 0, 0, 255, 255, 0, 255, j, i)
132 == 9)
133 {
134 pola += "16,";
135 rectangle(bmp1, j, i, 3, 3);
136 count++;
137 }
138 else if (cek_bif(255, 0, 255, 255, 0, 0, 0, 255, 255, j, i)
139 == 9)
```

```
{
pola += "17,";
rectangle(bmp1, j, i, 3, 3);
count++;
}
else if (cek_bif(255, 0, 255, 0, 0, 255, 255, 255, 0, j, i)
== 9)
{
pola += "18,";
rectangle(bmp1, j, i, 3, 3);
count++;
}
else if (cek_bif(0, 255, 255, 255, 0, 0, 255, 0, 255, j, i)
== 9)
{
pola += "19,";
rectangle(bmp1, j, i, 3, 3);
count++;
}
else if (cek_bif(255, 0, 255, 255, 255, 0, 0, 0, 255, j, i)
== 9)
{
pola += "20,";
rectangle(bmp1, j, i, 3, 3);
count++;
}
else if (cek_bif(0, 0, 255, 255, 255, 0, 255, 0, 255, j, i)
== 9)
{
pola += "21,";
rectangle(bmp1, j, i, 3, 3);
count++;
}
else if (cek_bif(255, 0, 0, 0, 255, 255, 255, 0, 255, j, i)
== 9)
{
pola += "22,";
rectangle(bmp1, j, i, 3, 3);
count++;
}
else if (cek_bif(255, 0, 255, 0, 255, 255, 255, 0, 0, j, i)
== 9)
{
pola += "23,";
rectangle(bmp1, j, i, 3, 3);
count++;
}
}
pola = pola.Substring(0, pola.Length - 1);
database(pola);
box1_img_tes.Image = (Image)bmp1;
}

private int cek_bif(int b1, int b2, int b3, int b4, int b5,
int b6, int b7, int b8, int b9, int x, int y)
{
int benar = 0;
```

```
197  if (red[x - 1, y - 1] == b1)
198  benar++;
199  if (red[x, y - 1] == b2)
200  benar++;
201  if (red[x + 1, y - 1] == b3)
202  benar++;
203  if (red[x - 1, y] == b4)
204  benar++;
205  if (red[x, y] == b5)
206  benar++;
207  if (red[x + 1, y] == b6)
208  benar++;
209  if (red[x - 1, y + 1] == b7)
210  benar++;
211  if (red[x, y + 1] == b8)
212  benar++;
213  if (red[x + 1, y + 1] == b9)
214  benar++;
215  return benar;
216  }
217
218  private Bitmap rectangle(Bitmap bmp, int x, int y, int
219  jarakx, int jaraky)
220  {
221  int batasy1 = y - jaraky;
222  int batasy2 = y + jaraky;
223  int batasx1 = x - jarakx;
224  int batasx2 = x + jarakx;
225
226  if (batasx1 < 0) batasx1 = 0;
227  if (batasx2 > bmp.Width) batasx2 = bmp.Width;
228  if (batasy1 < 0) batasy1 = 0;
229  if (batasy2 > bmp.Height) batasy2 = bmp.Height;
230
231  for (int i = batasy1; i < batasy2; i++)
232  for (int j = batasx1; j < batasx2; j++)
233                  {
234  if (i == batasy1 || i == batasy2 - 1 || j == batasx1 || j ==
235  batasx2 - 1)
236  bmp.SetPixel(j, i, Color.Red);
237  }
238  return bmp;
239  }
```

Penjelasan mengenai *Sourcecode* 4.4 proses ekstraksi fitur sebagai berikut :

Baris 3-13 adalah memasukkan nilai piksel ke dalam array

Baris 15-193 adalah pengecekan *bifurcation* yang ada pada citra dan memasukkan nama simbol ke dalam *database*.

Baris 195-218 adalah menginisialisasikan 8-ketetanggaan

Baris 220-241 adalah proses pemberian tanda merah pada citra yang terdapat *bifurcation.*

4.3.5 Implementasi Parameter HMM

Dalam proses parameter HMM meliputi matriks emisi, matriks transisi, matriks inisial, proses algoritma *forward*, algoritma *backward*, matriks *gamma*, matriks *epsilon*, evaluasi hasil, dan re-estimasi parameter HMM.

4.3.5.1 Implementasi Matriks Emisi

Matriks emisi digunakan untuk menjalankan algoritma *forward* dan *backward* baik proses inisialisasi maupun induksi. *Sourcecode* 4.5 adalah matriks emisi ditunjukkan sebagai berikut :

Sourcecode 4.5 matriks emisi

```
1   public HiddenMarkovModel(int symbols, int states)
2   : this(symbols, states, HiddenMarkovModelType.Ergodic)
3   {
4   }
5
6   public HiddenMarkovModel(int symbols, int states,
7   HiddenMarkovModelType type)
8   : this(null, null, states, type)
9   {
10  if (symbols <= 0)
11  {
12  throw new ArgumentOutOfRangeException("symbols", "Number of
13  symbols should be higher than zero.");
14  }
15
16  this.symbols = symbols;
17  this.B = new double[states, symbols];
18
19  for (int i = 0; i < states; i++)
20  for (int j = 0; j < symbols; j++)
21  B[i, j] = 1.0 / symbols;
22  }
```

Penjelasan *Sourcecode* 4.5 implementasi matriks emisi sebagai berikut :

Baris 1-5 adalah konstruktor dari fungsi.

Baris 7-25 adalah pengecekan simbol kurang dari 0 atau tidak dan menghitung nilai matriks emisi.

4.3.5.2 Implementasi Matriks Transisi Dan Inisial

Matriks transisi dan inisial juga digunakan untuk menjalankan algoritma *forward* dan *backward* baik proses inisialisasi maupun induksi. *Sourcecode* 4.6 adalah matriks transisi dan inisial ditunjukkan sebagai berikut :

Sourcecode 4.6 matriks transisi dan inisial

```
1   private HiddenMarkovModel(double[,] transitions, double[]
2   probabilities, int? states, HiddenMarkovModelType type)
3   {
4   #region Number of states N
5
6   int n = states.Value;
7   this.states = n;
8
9   #endregion
10
11  #region Transitions Matrix A
12
13  if (type == HiddenMarkovModelType.Ergodic)
14  {
15  transitions = new double[n, n];
16  for (int i = 0; i < n; i++)
17  for (int j = 0; j < n; j++)
18  transitions[i, j] = 1.0 / n;
19  }
20
21  this.A = transitions;
22
23  #endregion
24
25  #region Initial Probabilities pi
26
27  if (probabilities == null)
28  {
29  probabilities = new double[n];
30  probabilities[0] = 1.0;
31  }
32  this.pi = probabilities;
33
34  #endregion
35
36  }
```

Penjelasan *Sourcecode* 4.6 implementasi matriks transisi dan inisial sebagai berikut :

Baris 6-7 adalah menentukan nilai n adalah *state*.

Baris 13-21 adalah menghitung nilai matriks transisi.

Baris 27-32 adalah menentukan nilai matriks inisial.

4.3.5.3 Implementasi Algoritma *Forward*

Algoritma *forward* digunakan untuk membentuk matriks *forward* yang digunakan dalam menghitung matriks *epsilon* dan *gamma*. *Sourcecode* 4.7 adalah algoritma *forward* yang ditunjukkan sebagai berikut :

Sourcecode 4.7 Algoritma *forward*

```
1   private double[,] forward(int[] observations, out double[]
```

```
2     c)
3     {
4     int T = observations.Length;
5     double[] pi = Probabilities;
6     double[,] A = Transitions;
7
8     double[,] fwd = new double[T, States];
9     c = new double[T];
10
11    // 1. Initialization
12    for (int i = 0; i < States; i++)
13    c[0] += fwd[0, i] = pi[i] * B[i, observations[0]];
14
15    if (c[0] != 0) // Scaling
16    {
17    for (int i = 0; i < States; i++)
18    fwd[0, i] = fwd[0, i] / c[0];
19    }
20
21    // 2. Induction
22    for (int t = 1; t < T; t++)
23    {
24    for (int i = 0; i < States; i++)
25    {
26    double p = B[i, observations[t]];
27    double sum = 0.0;
28    for (int j = 0; j < States; j++)
29    sum += fwd[t - 1, j] * A[j, i];
30    fwd[t, i] = sum * p;
31
32    c[t] += fwd[t, i]; // scaling coefficient
33    }
34
35    if (c[t] != 0) // Scaling
36    {
37    for (int i = 0; i < States; i++)
38    fwd[t, i] = fwd[t, i] / c[t];
39    }
40    }
41
42    return fwd;
43    }
```

Penjelasan *Sourcecode* 4.7 implementasi algoritma *forward* sebagai berikut :

Baris 3-8 adalah menentukan batasan dan variable yang dipakai dalam algoritma *forward*.

Baris 12-20 adalah proses inisialisasi dan normalisasi hasil inisialisasi dari algoritma *forward*.

Baris 24-43 adalah proses induksi dan normalisasi hasil induksi dari algoritma *forward*.

4.3.5.4 Implementasi Algoritma *Backward*

Sama halnya dengan algoritma *forward*, algoritma *backward* juga digunakan untuk menghitung nilai dari matriks *gamma* dan *epsilon*. *Sourcecode* 4.8 untuk algoritma *backward* ditunjukkan sebagai berikut :

Sourcecode 4.8 Algoritma *backward*

```
1   private double[,] backward(int[] observations, double[] c)
2   {
3   int T = observations.Length;
4   double[] pi = Probabilities;
5   double[,] A = Transitions;
6
7   double[,] bwd = new double[T, States];
8
9   // 1. Initialization
10  for (int i = 0; i < States; i++)
11  bwd[T - 1, i] = 1.0 / c[T - 1];
12
13  // 2. Induction
14  for (int t = T - 2; t >= 0; t--)
15  {
16  for (int i = 0; i < States; i++)
17  {
18  double sum = 0;
19  for (int j = 0; j < States; j++)
20  sum += A[i, j] * B[j, observations[t + 1]] * bwd[t + 1, j];
21  bwd[t, i] += sum / c[t];
22  }
23  }
24
25  return bwd;
26  }
```

Penjelasan *Sourcecode* 4.8 implementasi algoritma *backward* sebagai berikut :

Baris 3-7 adalah variabel dan memberi batasan yang dipakai dalam algoritma *backward*.

Baris 9-11 adalah menghitung nilai inisialisasi dari algoritma *backward*.

Baris 14-24 adalah menghitung nilai induksi dari algoritma *backward*.

4.3.5.5 Implementasi Nilai Matriks *Gamma*

Nilai matriks *gamma* digunakan untuk estimasi kembali parameter HMM. *Sourcecode* 4.9 untuk menghitung nilai matriks *gamma* ditunjukkan sebagai berikut :

Sourcecode 4.9 Implementasi nilai matriks *gamma*

```
1   for (int t = 0; t < T; t++)
2   {
```

```
3     double s = 0;
4
5     for (int k = 0; k < States; k++)
6     s += gamma[i][t, k] = fwd[t, k] * bwd[t, k];
7
8     if (s != 0) // Scaling
9     {
10    for (int k = 0; k < States; k++)
11    gamma[i][t, k] /= s;
12    }
13    }
```

Penjelasan *Sourcecode* 4.9 implementasi nilai matriks *gamma* sebagai berikut :

Baris 6 adalah proses perhitungan nilai matriks *gamma*.

Baris 12 adalah proses perhitungan normalisasi nilai matriks *gamma*.

4.3.5.6 Implementasi Nilai Matriks *Epsilon*

Sama halnya dengan nilai matriks *gamma*, nilai matriks *epsilon* juga digunakan untuk estimasi kembali parameter HMM. *Sourcecode* 4.10 untuk menghitung nilai matriks *epsilon* sebagai berikut :

Sourcecode 4.10 Implementasi nilai matriks *epsilon*

```
1     // Calculate epsilon values for next computations
2     for (int t = 0; t < T - 1; t++)
3     {
4     double s = 0;
5
6     for (int k = 0; k < States; k++)
7     for (int l = 0; l < States; l++)
8     s += epsilon[i][t, k, l] = fwd[t, k] * A[k, l] * bwd[t + 1,
9     l] * B[l, sequence[t + 1]];
10
11    if (s != 0) // Scaling
12    {
13    for (int k = 0; k < States; k++)
14    for (int l = 0; l < States; l++)
15    epsilon[i][t, k, l] /= s;
16    }
17    }
```

Penjelasan *Sourcecode* 4.10 implementasi nilai matriks *epsilon* sebagai berikut :

Baris 8-9 adalah proses perhitungan nilai matriks *epsilon*.

Baris 16 adalah proses normalisasi nilai matriks *epsilon*.

4.3.5.7 Implementasi Evaluasi Hasil

Jika kondisi konvergen tercapai maka hasil *likelihood* akan dievaluasi. *Sourcecode* 4.11 untuk mengevaluasi hasil ditunjukkan sebagai berikut :

Sourcecode 4.11 Implementasi evaluasi hasil

```
1   public double Evaluate(int[] observations, bool logarithm)
2   {
3   if (observations == null)
4   throw new ArgumentNullException("observations");
5
6   if (observations.Length == 0)
7   return 0.0;
8
9
10  // Forward algorithm
11  double likelihood = 0;
12  double[] coefficients;
13
14  // Compute forward probabilities
15  forward(observations, out coefficients);
16
17  for (int i = 0; i < coefficients.Length; i++)
18  likelihood += Math.Log10(coefficients[i]);
19
20  // Return the sequence probability
21  return logarithm ? likelihood : Math.Exp(likelihood);
22  }
```

Penjelasan *Sourcecode* 4.11 implementasi evaluasi hasil sebagai berikut :

Baris 3-8 adalah proses pengecekan observasi dan panjang observasi.

Baris 12-16 adalah menginisialisasikan variabel dan mendapatkan nilai koefisien dari matriks *forward*.

Baris 18-23 adalah perhitungan nilai evaluasi dan pengecekan nilai *likelihood* yang diperoleh dari hasil evaluasi.

4.3.5.8 Implementasi Re-estimasi Nilai Matriks Inisial

Reestimasi matriks inisial bertujuan untuk menghitung kembali nilai matriks *forward* dan *backward* yang belum mencapai kondisi konvergen. *Sourcecode* 4.12 untuk re-estimasi matriks inisial ditunjukkan sebagai berikut :

Sourcecode 4.12 Implementasi re-estimasi nilai matriks inisial

```
1   // 3.1 Re-estimation of initial state probabilities
2   for (int k = 0; k < States; k++)
3   {
4   double sum = 0;
5   for (int i = 0; i < N; i++)
6   sum += gamma[i][0, k];
```

```
7    pi[k] = sum / N;
8    }
```

Penjelasan *Sourcecode* 4.12 implementasi re-estimasi nilai matriks inisial sebagai berikut :

Baris 1-8 adalah proses menentukan nilai matriks inisial.

4.3.5.9 Implementasi Re-estimasi Nilai Matriks Transisi

Sama halnya dengan matrik inisial, matriks transisi diestimasi kembali untuk menghitung nilai matriks *forward* dan *backward* agar mencapai kondisi konvergen. *Sourcecode* 4.13 untuk re-estimasi matriks transisi ditunjukkan sebagai berikut :

Sourcecode 4.13 Implementasi reestimasi nilai matriks transisi

```
1    // 3.2 Re-estimation of transition probabilities
2    for (int i = 0; i < States; i++)
3    {
4    for (int j = 0; j < States; j++)
5    {
6    double den = 0, num = 0;
7
8    for (int k = 0; k < N; k++)
9    {
10   int T = observations[k].Length;
11
12   for (int l = 0; l < T - 1; l++)
13   num += epsilon[k][l, i, j];
14
15   for (int l = 0; l < T - 1; l++)
16   den += gamma[k][l, i];
17   }
18
19   A[i, j] = (den != 0) ? num / den : 0.0;
20   }
21   }
```

Penjelasan *Sourcecode* 4.13 implementasi re-estimasi nilai matriks transisi sebagai berikut :

Baris 2-16 adalah proses perhitungan nilai num dan den.

Baris 19-20 adalah proses pengecekan nilai den dan proses menentukan nilai matriks transisi.

4.3.5.10Implementasi Re-estimasi Nilai Matriks Emisi

Sama halnya dengan matriks inisial dan transisi, matriks emisi diestimasi kembali untuk menghitung nilai matriks *forward* dan *backward* agar mencapai

kondisi konvergen. *Sourcecode* 4.14 untuk reestimasi nilai matriks emisi ditunjukkan sebagai berikut :

Sourcecode 4.14 Implementasi reestimasi nilai matriks emisi

```
1   for (int i = 0; i < States; i++)
2   {
3   for (int j = 0; j < Symbols; j++)
4   {
5   double den = 0, num = 0;
6
7   for (int k = 0; k < N; k++)
8   {
9   int T = observations[k].Length;
10
11  for (int l = 0; l < T; l++)
12  {
13  if (observations[k][l] == j)
14  num += gamma[k][l, i];
15  }
16
17  for (int l = 0; l < T; l++)
18  den += gamma[k][l, i];
19  }
20
21  B[i, j] = (num == 0) ? 1e-10 : num / den;
22  }
23  }
```

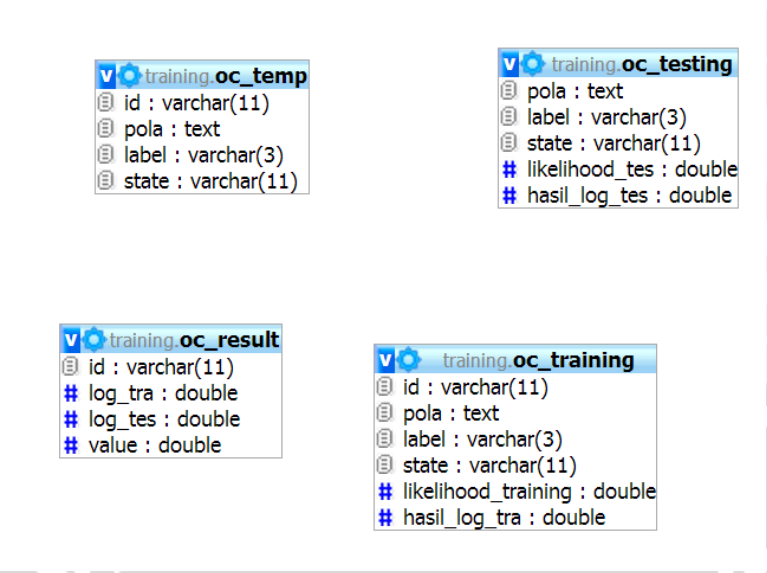
Penjelasan *Sourcecode* 4.14 implementasi re-estimasi nilai matriks emisi sebgai berikut :

Baris 1-20 adalah proses perhitungan nilai den dan num serta proses pengecekan nilai observasi pada kolom k dan baris l sesuai dengan nilai j.

Baris 22-23 adalah pengecekan nilai num dan proses menentukan nilai matriks emisi.

4.4 Implementasi Tabel Dalam Sistem Pengenalan Sidik Jari

Untuk implementasi database yang digunakan dalam sistem pengenalan sidik jari ditunjukkan pada Gambar 4.1 sebagai berikut :



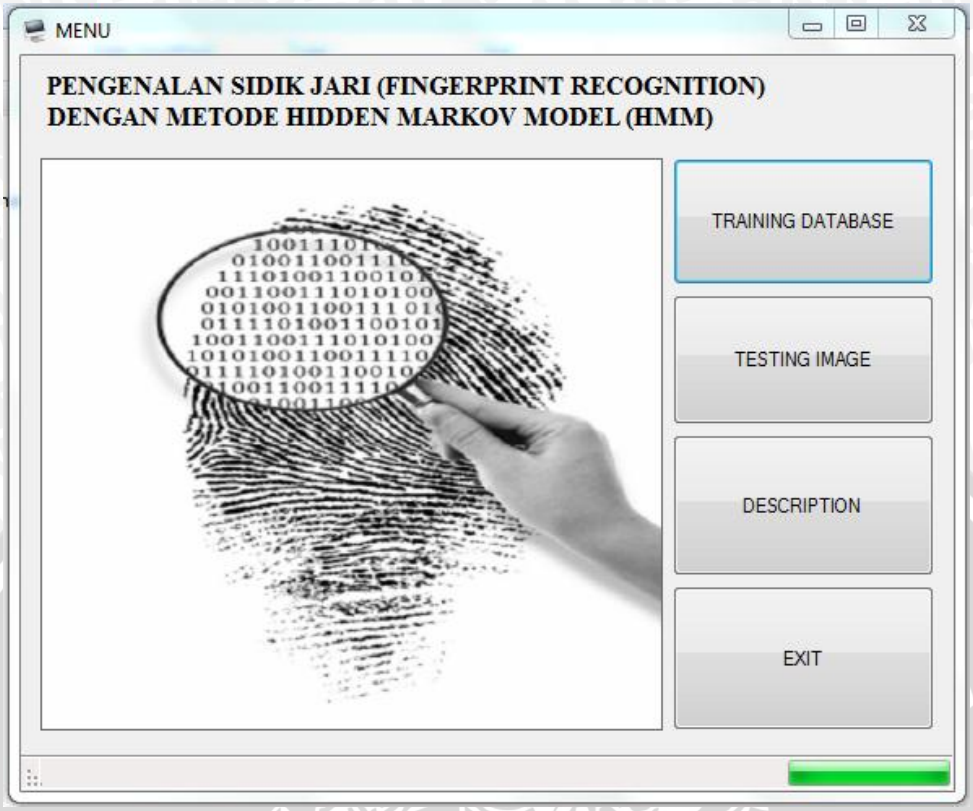
Gambar 4.1 Implementasi tabel dalam sistem

4.5 Implementasi Interface

Implementasi interface terdiri dari 4 bagian utama, yaitu :

a. Form utama

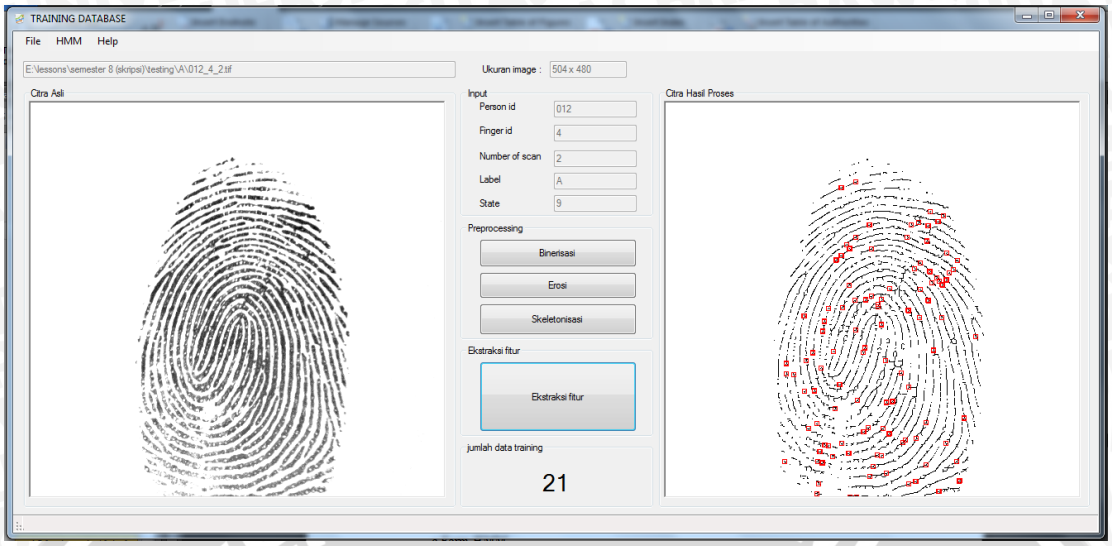
Form ini menampilkan seluruh *menu* yang ada pada pengenalan sidik jari. Adapun *menu* dalam form ini antara lain : *menu training database* yang digunakan untuk melatih citra, *menu testing image* digunakan untuk mengetahui hasil citra yang diuji, *menu description* berisi penjelasan singkat mengenai pengenalan sidik jari dengan metode *hidden markov model* (HMM), dan *menu exit* yang digunakan untuk keluar dari aplikasi. Adapun tampilannya ditunjukkan pada Gambar 4.2 berikut ini :



Gambar 4.2 Implementasi form utama

b. Form *Training*

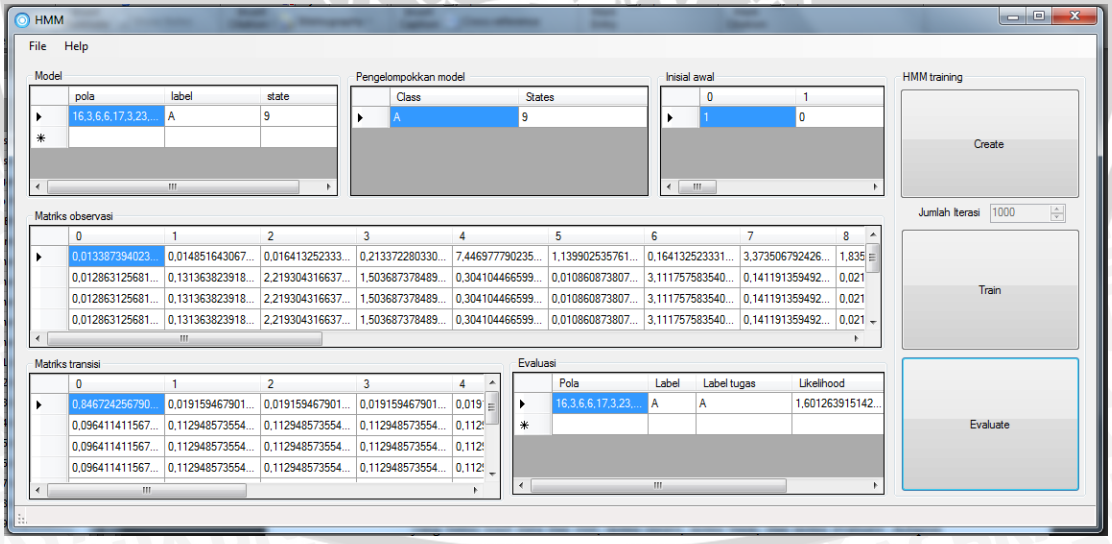
Form ini memasukkan hasil ekstraksi fitur yang ditemukan, label, *state.* dan id dari citra yang dilatih ke dalam *database*. Adapun *menu* yang terdapat dalam form *training* adalah *menu file* yang didalamnya terdapat *menu open* dan *exit*, *menu* HMM yang didalamnya berisi parameter dan hapus *database*. *Menu* binerisasi, *menu* erosi, *menu* skeletonisasi, *menu* ekstraksi fitur. Adapun tampilan form *training* ditunjukkan pada Gambar 4.3 berikut ini :



Gambar 4.3 Implementasi form *training*

c. Form HMM

Form ini memasukkan menghitung parameter HMM yang ada pada citra dan menghitung nilai *likelihood*. *Menu* yang terdapat dalam form ini adalah *menu file* yang berisi *load* data dan *exit*, *menu create*, *menu train*, dan *menu evaluate*. Adapun tampilan form HMM ditunjukkan pada Gambar 4.4 berikut ini :

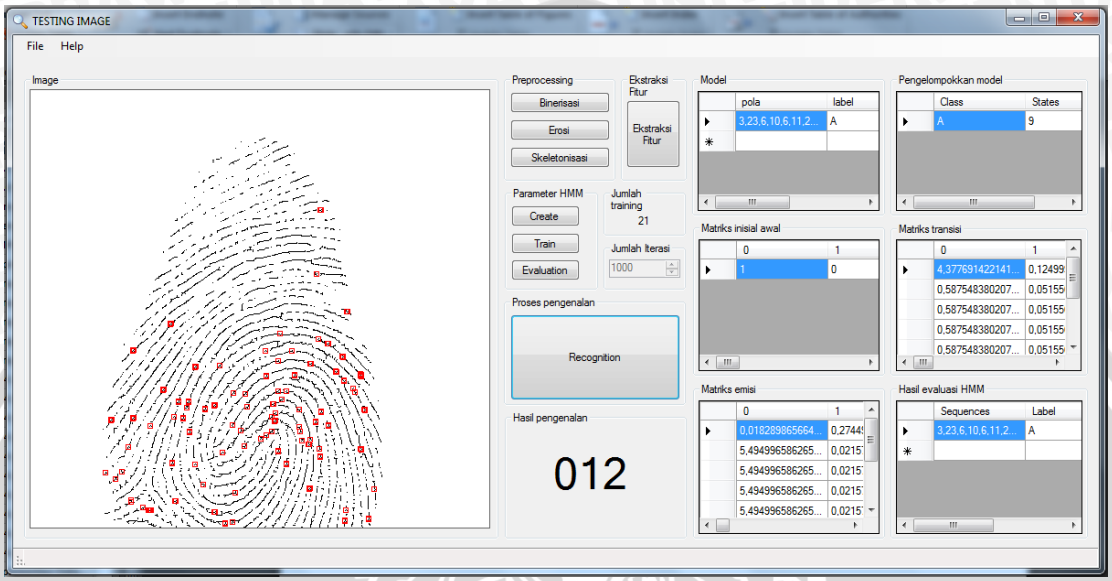


Gambar 4.4 Implementasi form HMM

d. Form *Testing*

Form ini memasukkan citra yang akan diuji dan hasil pengenalannya sama dengan id dikelasnya atau tidak. *Menu* yang terdapat pada form ini adalah *menu*

file yang berisi *open*, *load data*, dan *exit*, *menu* binerisasi, *menu* erosi, *menu* skeletonisasi, *menu* ekstraksi fitur, *menu* create, *menu* train, *menu* evaluate, dan *menu* recognition. Adapun tampilan form ditunjukkan pada Gambar 4.5 berikut ini :



Gambar 4.5 Implementasi form *testing*