BAB III IMPLEMENTASI SISTEM

Pada bab ini akan dibahas mengenai perangkat lunak yang akan dibuat. Bab ini akan menjelaskan mengenai spesifikasi sistem, alur program dan implementasinya pada perangkat lunak.

3.1 Spesifikasi Sistem

Pada bagian spesifikasi sistem ini akan dijelaskan mengenai komponen–komponen yang digunakan untuk membangun sistem. Komponen–komponen yang dijelaskan meliputi komponen perangkat keras dan perangkat lunak.

3.1.1 Perangkat Keras

Pada uji coba, perangkat lunak ini akan dijalankan pada *notebook* dengan spesifikasi perangkat keras sebagai berikut:

- Prosesor: Intel®Core™2 Duo T5550 (1.83 GHz, 667 MHz FSB, 2 MB L2 cache)
- Memori: 1.5 GB DDR2 PC-5300
- Harddisk: 160 GB

3.1.2 Perangkat Lunak

Adapun perangkat lunak yang digunakan pada saat implementasi program adalah sebagai berikut:

- Sistem Operasi: Windows XP SP 3 Professional Edition
- IDE: Microsoft Visual Studio 2005 Professional Edition dan MATLAB R2008a
- Bahasa Pemrograman: C# dan MATLAB
- Framework: .NET 2.0

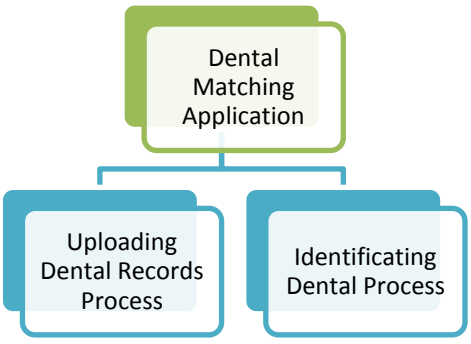
3.2 Rancangan Sistem

Pada perangkat lunak ini, antar muka yang dibuat akan dirancang semudah mungkin pada sisi navigasi dan dibuat dengan tampilan menarik. Perancangan

akan disesuaikan dengan kebutuhan yang ada serta sesuai dengan hasil analisis kebutuhan sistem yang telah dilakukan. Pada pengembangan navigasi, antarmuka akan dibuat mengikuti prinsip-prinsip sistem interaksi agar dapat menghasilkan sebuah perangkat lunak yang *user friendly*.

Gambar 3. 1 Tampilan Sistem

Perangkat lunak ini memiliki 2 fitur yaitu proses identifikasi dan proses *uploading* data ke basis data dalam bentuk berkas xml. Pada proses identifikasi, pengguna harus memilih *dental x-ray* hasil pemindaian dalam format JPEG. Format ini dipilih karena format JPEG telah menjadi format umum pada sebuah citra.

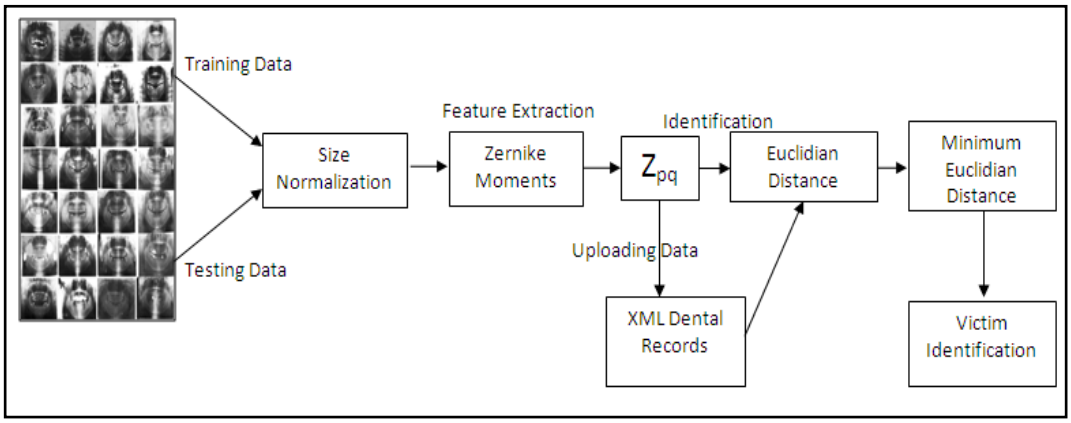


Gambar 3. 2 Dental matching Proses

Masukan utama pada perangkat lunak ini adalah citra *grayscale*. Namun, perangkat lunak ini juga memfasilitasi pengkonversian citra dari RGB image menjadi *grayscale* image. Citra RGB akan otomatis dikonversi menjadi *grayscale* pada saat pemrosesan data.

Pada penghitungan *Zernike moments* suatu citra, ukuran dari citra tersebut harus dinormalisasi sehingga ukuran gambar tersebut menjadi NxN. Pada perangkat lunak ini disediakan juga fitur otomasi mengubah ukuran citra menjadi 128x128.

Proses *uploading* data merupakan proses menyimpan data hasil ekstraksi citra *dental x-ray* serta memasukan identitas dari pemilik dental tersebut. Hasil ekstraksi dan data identitas pemilik tersebut akan disimpan pada berkas xml yang nantinya akan dicocokan pada proses identifikasi.



Gambar 3. 3 Alur Sistem Keseluruhan

Pada proses identifikasi *dental x-ray* yaitu dengan cara melakukan ekstraksi ciri dari citra dental dengan menghitung nilai *Zernike moment*s. Pada perangkat lunak ini, pengguna dapat mem-*browse* citra dental yang telah ada, kemudian memilih *order moment* yang diinginkan.

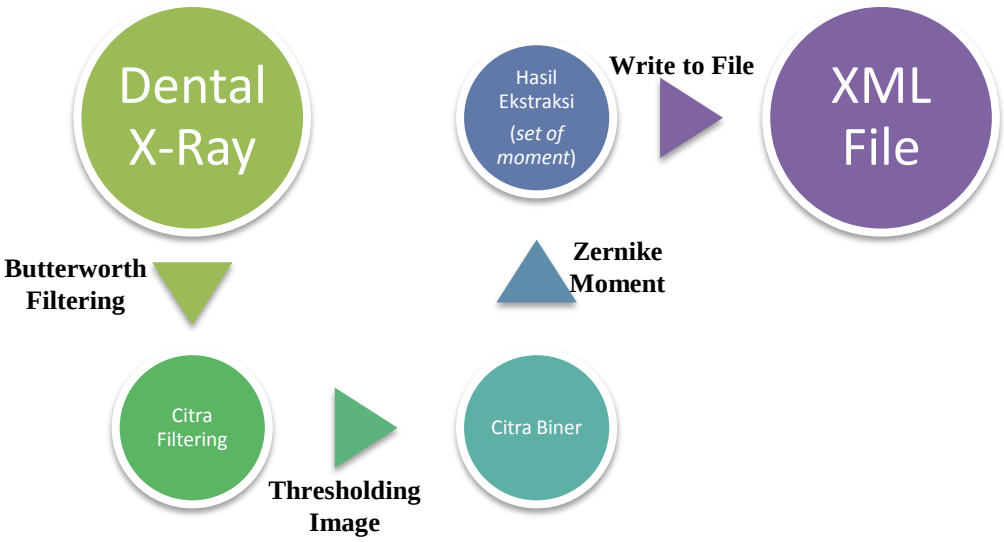
Tahapan selanjutnya, nilai *Zernike moment*s yang ditemukan pada proses identifikasi akan dicocokan dengan basis data dental (*ante mortem dental records*) dengan menggunakan *euclidian distance*. Jarak euclid yang terpendek akan menjadi pemenang pada proses identifikasi ini.

3.3 Proses Uploading Data

Tahapan *uploading data* merupakan salah satu fitur yang ada pada perangkat lunak ini. Pada proses ini, data citra *dental x-ray* diekstraksi lalu akan disimpan dalam berkas XML. Pada perangkat lunak ini *order of moment* dibatasi sampai 40, sehingga ketika *uploading data* terjadi citra dental tersebut akan dihitung hingga order ke -40.

Pada proses ini data pemilik citra *dental x-ray* akan di-*upload* bersamaan dengan *uploading* data dental. Data pribadi juga akan disimpan pada berkas XML. Hasil dari proses *uploading data* ini adalah berupa *dental records* dalam bentuk berkas xml.

Berikut alur program pada saat proses *uploading data* untuk masukan jenis citra *grayscale*.



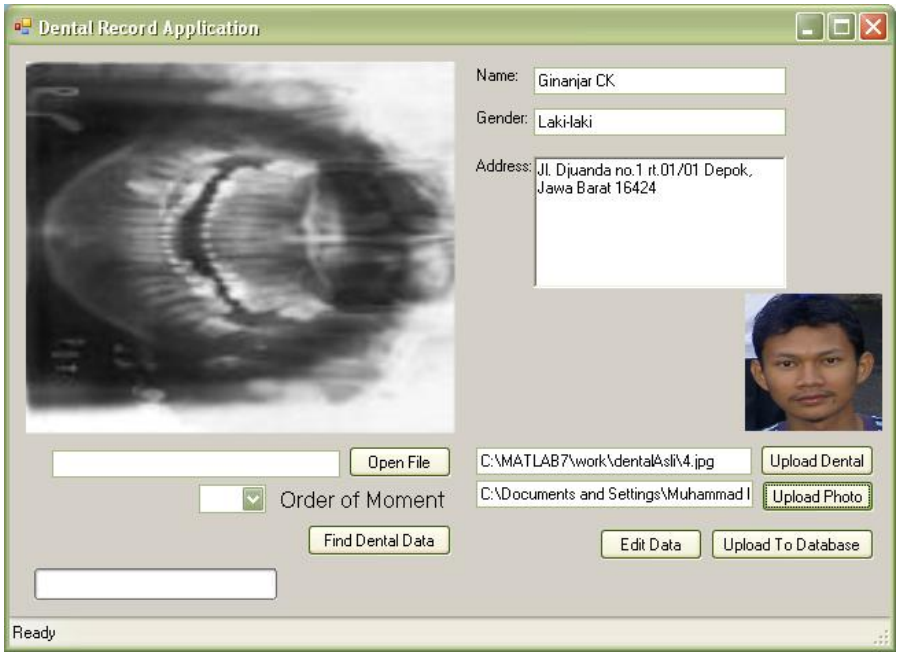
Gambar 3. 4 Uploading Binary Data Process

Pada tahapan awal, citra akan di-*filter* menggunakan *high-pass butterworth filtering*. *High-pass filtering* ini digunakan untuk lebih memperlihatkan dan mempertajam bentuk (*edge*) dari dental pada citra *dental x-ray* tersebut. *High-pass filtering* akan memudahkan pada proses binerisasi citra, sebab bentuk dari gigi lebih terlihat dengan menggunakan teknik *filtering* ini.

Setelah citra hasil *filtering* didapatkan, maka tahapan selanjutnya adalah proses binerisasi citra. Proses binerisasi ini merupakan proses *thresholding* pada citra *grayscale* hasil *filtering*. Citra biner ini merupakan sebuah citra yang hanya memperlihatkan nilai 0 atau 1, dengan kata lain adalah citra hitam putih.

Penelitian ini juga akan memperlihatkan apakah proses binerisasi citra ini dapat meningkatkan tingkat pengenalan perangkat lunak yang sedang dikembangkan. Citra biner akan memperjelas bentuk dari citra dental. Namun, permasalahan terletak pada pembatasan *threshold* dan *cutoff* sebab pembatasan *threshold* dan *cutoff* yang terlalu tinggi dapat menyebabkan bentuk dental tidak terlihat, dan juga sebaliknya *threshold* dan *cutoff* yang terlalu rendah dapat memburamkan bentuk citra dental tersebut.

Sistem juga akan diuji menggunakan masukan citra grayscale tanpa ada proses *filtering*. Tujuan digunakannya citra *grayscale* dan biner adalah untuk melihatkan citra yang cocok untuk digunakan pada perangkat lunak yang dikembangkan ini. *Zernike moments* terkenal menjadi sebuah alat *descriptor* yang *powerfull* pada citra *grayscale* dan biner. Dari hasil uji coba dan analisis maka akan ditentukan hasil terbaik pada penggunaan citra *grayscale* atau biner.



Gambar 3. 5 Screenshot Proses Uploading Data

Setelah proses *filtering* yang akan menghasilkan citra biner, maka perangkat lunak akan menghitung secara otomatis nilai *Zernike moments* dari citra tersebut hingga order 40. Order 40 ini dipilih karena dirasakan bisa merepresentasikan informasi yang diperlukan pada citra dental. Selain itu, pengambilan *order moment* yang terlalu tinggi justru dapat merusak tingat pengenalan pada citra dental.

Penggunaan *order moment* yang tinggi dapat memperoleh informasi yang lebih detail dari citra dental. Namun, *noise* yang banyak pada citra dental justru dapat mengganggu informasi yang diproses. Sebab pada order tinggi, *noise* yang ada pada citra dental juga akan dikenali sebagai informasi yang diperlukan.

Gambar 3. 6 Uploading *Grayscale* Data Process

Pada proses uploading data untuk citra *grayscale* cukup sederhana, karena tidak melalui proses *filtering* dan proses binerisasi citra. Proses ini cukup mengekstraksi ciri dari citra dental *grayscale*. Proses yang terdapat pada jenis masukan ini hanya melakukan ekstraksi citra menggunakan *Zernike moments* untuk citra *grayscale*.

Penggunaan 2 jenis masukan yang berbeda ini yaitu penghitungan Zernike moment pada citra biner dan penghitungan Zernike moment pada citra *grayscale*. Hal ini akan digunakan sebagai bahan acuan untuk melihat jenis masukan yang lebih baik antara kedua masukan ini.

3.4 Proses Identifikasi

Tahapan ini merupakan tahapan identifikasi sebuah dental untuk mengetahui identitas korban. Tahapan ini mencocokan kondisi *post mortem* (kondisi dental korban) dengan *ante mortem* (*dental records* sebelum meninggal). Pada perangkat

lunak ini, data *ante mortem* merupakan data dental yang telah di-*upload* ke basis data. Proses *uploading* data telah dibahas pada subbab sebelumnya.

Proses identifikasi ini membandingkan ekstraksi ciri dari dental korban dengan basis data yang didapatkan pada proses uploading data. Proses kesamaan akan menggunakan algortima jarak Euclid. Jarak yang menghasilkan nilai minimum akan menjadi kandidat utama sebagai korban yang teridentifikasi.

Ada variable yang mempengaruhi proses identifikasi pada perangkat lunak ini yaitu *order* dari *moment*. *Order* dari *moment* dapat dipilih secara manual oleh pengguna. *Order of moment* merupakan tingat ketelitian informasi yang akan dihasilkan dari proses ekstraksi data. Informasi yang diperoleh dari setiap *order* bersifat *orthogonal*, maka dari itu semakin tinggi *order moment* digunakan semakin detail informasi dari suatu citra didapatkan.

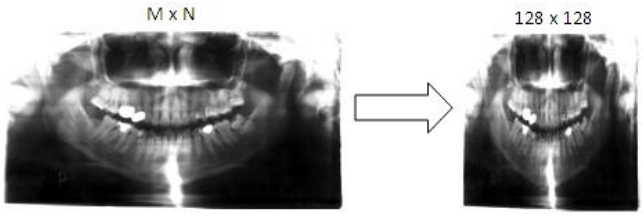
Namun, teori semakin tinggi *order moment* yang digunakan akan semakin meningkatkan ketelitian ekstraksi ciri yang dilakukan tidak berlaku pada citra dental *x-ray*. Pada citra dental *x-ray*, *noise* yang ada pada citra cukup banyak sehingga penggunaan *order moment* yang terlalu tinggi justru akan menurunkan tingkat pengenalan disebabkan gangguan *noise* yang ada.

Pada perangkat lunak ini, *order* dari *moment* dibatasi dari 1 sampai 40 sebab keterbatasan komputasi yang dibutuhkan untuk menghitung *moment* pada *order* yang tinggi. Selain itu pada order 40, informasi yang dihasilkan cukup akurat. Pemilihan order 40 dilakukan sebab pada order yang lebih tinggi citra *dental x-ray* yang memiliki banyak noise justru akan semakin menurunkan tingkat pengenalan citra.

Order yang tinggi akan menghasilkan informasi yang lebih detail namun tak tertutup kemungkinan informasi yang dihasilkan adalah *noise* pada citra tersebut. Oleh karena itu, pembatasan *order* dari *moment* harus dilakukan karena beban komputasi, dan juga dapat meningkatkan pengenalan.

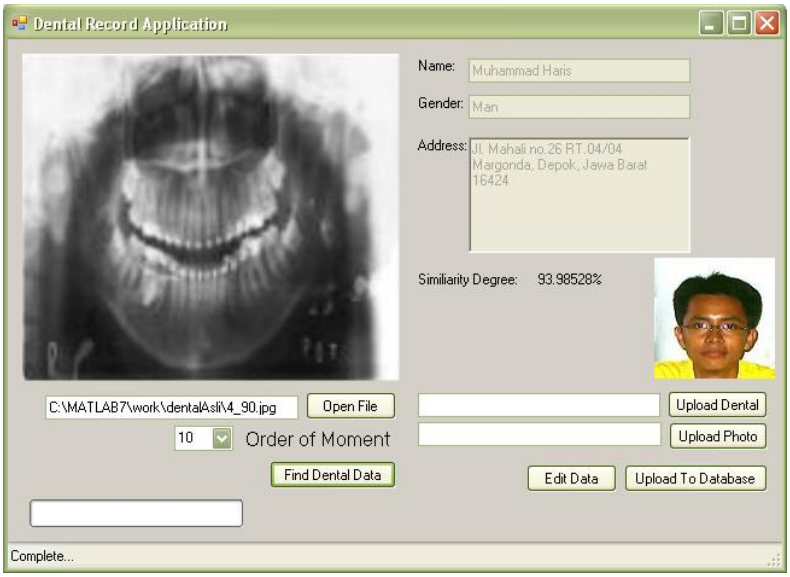
Proses ini juga akan melakukan pemeriksaan pada citra dental yang akan diekstraksi. Citra yang akan diekstraksi harus berupa citra *grayscale*. Namun, bila pengguna menggunakan citra RGB, perangkat lunak ini akan secara otomatis melakukan konversi citra dari citra RGB ke citra *grayscale*.

Tahapan ini juga akan melakukan normalisasi ukuran pada citra dental yaitu sesuai standar sistem dengan ukuran 128x128. Standar ini dibuat untuk mengurangi beban komputasi yang akan dilakukan sebab bila ukuran terlalu besar, proses komputasi akan memakan waktu yang lama.



Gambar 3. 7 Normalisasi ukuran citra

Proses identifikasi diawali dengan memilih citra dental *post mortem* dari korban yang telah dilakukan pemindaian citra *dental x-ray*. Setelah itu, pengguna memilih *order of moment* dari penghitungan yang akan dilakukan. Berikut akan diberikan *screenshot* dari proses identifikasi korban.



Gambar 3. 8 Tampilan Tahap Identifikasi

Proses identifikasi ini menggunakan *Euclidian distance* yang berfungsi untuk melihat jarak kedekatan antara masing-masing *dental records* dengan kondisi dental korban bencana. Proses ini menggunakan nilai absolut dari tiap nilai *Zernike moments* pada citra dental tersebut. Nilai absolut digunakan untuk memperoleh *rotation invariance* yang dimiliki oleh *Zernike moments* seperti yang ditunjukkan persamaan **(3.1)** dan **(3.2)** di bawah ini.

Nilai absolute atau *magnitude* dari *Zernike moments* ini sangat bermanfaat pada perangkat lunak ini sebab merupakan komponen utama untuk mengenali citra dental walaupun citra tersebut mengalami rotasi dan distorsi. Pernyataan ini akan dibuktikan pada bagian eksperimen karena skenario eksperimen akan melihat tingkat pengenalan sistem pada citra yang mengalami rotasi, dan distorsi.

Nilai *Zernike moment*s dari sebuah citra yang mengalami rotasi ($Z^r{}_{pq}$) mempunyai keterhubungan dengan citra asli yang belum mengalami rotasi [BIN02]. Keterhubungan tersebut dijelaskan pada persamaan di bawah ini:

$$\boldsymbol{Z^r{}_{pq} = Z_{pq} \exp(-jm\theta)} \qquad \textbf{(3. 1)}$$

$$\boldsymbol{|Z^r{}_{pq}| = |Z_{pq}|} \qquad \textbf{(3. 2)}$$

Dari persamaan di atas dapat dilihat bahwa citra hasil rotasi memiliki nilai absolut yang sama dengan citra asli. Oleh karena itu, dapat dilakukan penghitungan *Euclidian distance* sebagai berikut:

$$\boldsymbol{D = \sum_{(p,q)\in D} \sum (|Z_{pq}| - |Z'_{pq}|)^2} \qquad \textbf{(3. 3)}$$

Hasil pada proses identifikasi juga akan memperlihatkan tingkat kemiripan *post mortem* dan *ante mortem* dalam rasio persentase. Tingkat kemiripan diperlihatkan agar pengguna dapat melihat seberapa tingkat kemiripan yang dimiliki data *post mortem* dengan *ante mortem*.

Tingkat kemiripan diperoleh dari hasil perhitungan 1 dikurangi dengan hasil bagi nilai total *Zernike moments post mortem* dan nilai total *Zernike moments ante mortem* pemenang. Hasil dari perhitungan akan dikalikan 100% untuk

mendapatkan persentase tingkat kemiripan. $\boldsymbol{Z_t}$ merupakan nilai total Zernike moments post mortem, sedangkan $\boldsymbol{Z'_t}$ merupakan nilai total Zernike moment ante mortem pemenang. Berikut persamaan tingkat kemiripan yang dibuat oleh peneliti:

$$\mathbf{S = (1 - (Z_t - Z'_t)) \times 100\%} \qquad \mathbf{(3.\ 4)}$$

3.5 Implementasi Algoritma

Pada subbab ini akan dibahas mengenai implementasi algoritma ke dalam program C#. Bagian ini akan membahas implementasi komputasi *Zernike Moments*, penulisan dan pembacaan berkas xml, normalisasi ukuran citra, proses binerisasi, dan *Euclidian distance*.

3.5.1 Komputasi Zernike Moments

Implementasi algoritma *Zernike moments* memiliki 2 argumen input yaitu citra dental yang direpresentasikan menggunakan array 2 dimensi, sedangkan argumen kedua adalah kumpulan *order of moment* yang direpresentasikan menggunakan array 1 dimensi.

Method ini akan mengembalikan nilai *Zernike moments* pada citra dental sesuai dengan *order of moment* yang diberikan. Nilai *Zernike moments* akan berupa *complex number* yang disimpan pada sebuah array 1 dimensi.

Implementasi *Zernike moments* yang sebenarnya dilakukan pada Matlab, sehingga untuk memudahkan komunikasi antar bahasa antara C# dan Matlab akan digunakan *library .net builder* yang dimiliki Matlab. *Library* ini akan membuat .dll dari kode Matlab yang dibuat sehingga .dll tersebut akan menjadi salah satu *library* yang akan digunakan pada sistem pencocokan dental.

Pada bagian ini, kode yang ditampilkan hanya bagian pada C# untuk melihat proses kerja/alur dari perangkat lunak yang dikembangkan. Berikut merupakan implementasi dari komputasi *Zernike Moments*:

```
private Complex[] zernikeMoment(double[,] bnew, double[] ar) {
    progressBar1.PerformStep();
    zernikeclass AClass = new zernikeclass();
    MWNumericArray hasilFilter = new MWNumericArray(bnew);

    MWNumericArray iterasi = new MWNumericArray(ar);
    MWArray RetVal = AClass.zernike_moment(hasilFilter,
iterasi);
    Array cr =
((MWNumericArray)RetVal).ToVector(MWArrayComponent.Real);
    Array qr =
((MWNumericArray)RetVal).ToVector(MWArrayComponent.Imaginary);

    Complex[] result = new Complex[cr.GetLength(0)];

    for (int i = 0; i < cr.GetLength(0); i++) {
      result.SetValue(new
      Complex(Convert.ToDouble(cr.GetValue(i)),Convert.ToDouble
      (qr.GetValue(i))), i);
    }
    return result;
}
```

3.5.2 Penulisan dan Pembacaan Berkas XML

Setelah proses *uploading* data dilakukan, hasil penghitungan nilai *Zernike moments* yang mencapai order ke-40 akan disimpan pada berkas .xml. Selain itu pada proses identifikasi, berkas .xml yang disimpan pada saat *uploading* data akan dibaca pada proses ini. Oleh karena itu, pada bagian ini akan membahas sedikit tentang metode pembacaan dan penulisan pada berkas .xml.

Alasan pemilihan berkas .xml sebagai media penyimpanan adalah berkas ini sudah memiliki standar khusus sehingga penggunaan kembali pada data yang telah dihitung akan menjadi lebih mudah. Selain itu, format .xml merupakan format yang *human readable*. Hal ini akan memudahkan pada proses pengecekan data apabila terjadi kesalahan.

Ada 1 argumen yang diperlukan pada *method* yaitu nilai *Zernike moments* yang ingin disimpan. Nilai tersebut berupa *array* 1 dimensi, sehingga pada penulisan data akan dilakukan *looping* untuk menyimpan keseluruhan data.

Pada implementasi penulisan berkas .xml, *library class* yang digunakan adalah `XmlTextWriter`. *Library* ini biasa digunakan pada penulisan data dengan format

.xml. *Library* memudahkan pengguna untuk menuliskan data pada format .xml seperti penyediaan *method* `WriteStartDocument(),WriteStartElement()`.

Berikut merupakan implementasi dari penulisan berkas .xml:

```
private void writeXML(Complex[] zernikeMoment) {
      string s = Application.StartupPath + "\\DentalData\\";
      System.IO.DirectoryInfo d = new
System.IO.DirectoryInfo(s);
      int files = d.GetFiles().Length + 1;

      FileStream fs = new FileStream(s+files+".xml",
      FileMode.CreateNew, FileAccess.Write,
      FileShare.ReadWrite);
      XmlTextWriter w = new XmlTextWriter(fs, Encoding.UTF8);
      w.WriteStartDocument();
      w.WriteStartElement("Complex");
      w.WriteAttributeString("id", "" + files);
      for (int i = 0; i < zernikeMoment.Length; i++) {
       w.WriteStartElement("bilangan" + i);
       w.WriteAttributeString("real", zernikeMoment[i].Re +
"");
       w.WriteAttributeString("imaginary", zernikeMoment[i].Im
+ "");
       w.WriteEndElement();
      }
       w.WriteEndElement();
       w.WriteEndDocument();
       w.Flush();
       fs.Close();
}
```

Pada implementasi pembacaan berkas .xml, *library class* yang digunakan adalah `XmlTextReader`. *Library* ini biasa digunakan pada pembacaan data dengan format .xml. Library memudahkan pengguna untuk melakukan pembacaan pada data format .xml seperti penyediaan *method* `Read()`.

Berikut merupakan implementasi dari pembacaan berkas .xml:

```
private Complex[,] readXML(int order) {
      Complex[,] data = new Complex[d.GetFiles().Length,
jmlMoment];
      int iterasi = jmlMoment;
      for (int i = 1; i <= files; i++) {
            reader = new XmlTextReader(s+i+".xml");
            j=0;
            iterasi = jmlMoment;
       while (reader.Read()) {
       if (reader.NodeType.Equals(XmlNodeType.Element)){
       if (reader.Name.Equals("Complex")) {}
       else {
            double x =
Convert.ToDouble(reader.GetAttribute("real"));
                        double y =
            Convert.ToDouble(reader.GetAttribute("imaginary"));
            data[i - 1, j] = new Complex(x, y);
            j++;
            iterasi--;
            if (iterasi < 1) {
            break;
                  }
       }
       }
      }
            return data;
}
```

3.5.3 Normalisasi Ukuran Citra

Proses normalisasi digunakan untuk menyesuaikan ukuran citra yang diberikan agar sesuai standar yang telah peneliti tentukan. Ukuran standar yang dibuat adalah 128cm x 128cm. Ukuran ini dirasakan sudah dapat merepresentasikan kondisi citra dental dan memiliki komplexitas data yang cukup kecil. Pada tahapan implementasi normalisasi ukuran, ada 1 hal yang perlu diperhatikan yaitu perubahan ukuran tidak mengubah informasi yang terkandung pada suatu citra.

Pada implementasi digunakan interpolasi *biqubic* sehingga pemotongan pixel yang dilakukan dapat melihat dari keadaan pixel tetangganya. Metode interpolasi dapat menggunakan library yang ada pada *framework* .net 2.0 yaitu `InterpolationMode.HighQualityBicubic`. Berikut merupakan implementasi algoritma untuk melakukan normalisasi ukuran pada citra dental:

```
private static Image resizeImage(Image imgToResize, Size size)
{
   int sourceWidth = imgToResize.Width;
   int sourceHeight = imgToResize.Height;

   float nPercent = 0;
   float nPercentW = 0;
   float nPercentH = 0;

   nPercentW = ((float)size.Width / (float)sourceWidth);
   nPercentH = ((float)size.Height / (float)sourceHeight);

   if (nPercentH < nPercentW)
     nPercent = nPercentH;
   else
     nPercent = nPercentW;

   int destWidth = (int)(sourceWidth * nPercent);
   int destHeight = (int)(sourceHeight * nPercent);

   Bitmap b = new Bitmap(destWidth, destHeight);
   Graphics g = Graphics.FromImage((Image)b);
   g.InterpolationMode = InterpolationMode.HighQualityBicubic;

   g.DrawImage(imgToResize, 0, 0, destWidth, destHeight);
   g.Dispose();

   return (Image)b;
}
```

3.5.4 Proses Binerisasi

Tahapan binerisasi ini telah dijelaskan pada subbab sebelumnya. Implementasi ini sama seperti perhitungan nilai *Zernike moments* yaitu menggunakan Matlab. Setelah itu digunakan *.net builder library* untuk menghasilkan .dll yang dapat digunakan pada bahasa pemrograman C#. Berikut merupakan implementasi algoritma proses binerisasi dari citra *grayscale* ke citra biner:

```
private double[,] biner(double[,] data,int n) {
      zernikeclass AClass = new zernikeclass();
      MWNumericArray hasilbiner = new MWNumericArray(data);
      MWArray RetVal = AClass.biner(hasilbiner, n);

      Array temp =
      ((MWNumericArray)RetVal).ToVector(MWArrayComponent.Real);
      double[] cr = new double[temp.GetLength(0)];
      for (int i = 0; i < temp.GetLength(0); i++) {
            cr.SetValue(Convert.ToDouble(temp.GetValue(i)), i);
      }

      int tekj = cr.Length;

     int[] panjangArray = RetVal.Dimensions;
     int x = 0;
     double[,] result = new double[panjangArray[0],
panjangArray[1]];
     for (int i = 0; i < panjangArray[0]; i++) {
     for (int j = 0; j < panjangArray[1]; j++) {
        result.SetValue(Convert.ToDouble(cr.GetValue(x)),j,i);
        x++;
      }
      }
      return result;
}
```

3.5.5 Euclidian Distance

Implementasi jarak Euclid yang dibuat sama seperti perhitungan jarak Euclid pada umumnya. Namun, perbedaan terletak pada tipe data yang dihitung. Tipe data pada perhitungan ini adalah bilangan kompleks, sehingga perlu dihitung *magnitude* dari nilai *Zernike moments* tersebut.

Method ini memiliki 2 argumen yaitu nilai *Zernike moment post mortem* (keadaan gigi korban) dan nilai *Zernike moments* dari dental records. Nilai *Zernike moments* untuk *dental records* disimpan pada *array* 2 dimensi dengan tipe data bilangan kompleks. Nilai ini diperoleh setelah pembacaan berkas .xml hasil proses *uploading* data. Setelah itu, keduanya dibandingkan dengan mencari jarak Euclid antara *post mortem* dan masing-masing *ante mortem*.

Berikut merupakan implementasi algoritma Euclidian distance yang digunakan untuk menentukan tingat kesamaan sebuah citra:

```
private int euclidianDistance(Complex[] img, Complex[,]
databaseImg){
      int pemenang = 0;
      double[] totalData = new double[databaseImg.GetLength(0)];
      double temp,imgAbs,dataAbs;

      for (int i = 0; i < databaseImg.GetLength(0); i++ ) {
            temp = 0.0; imgAbs = 0.0; dataAbs = 0.0;
            for (int j = 0; j < img.Length; j++) {
            imgAbs = Math.Pow(Math.Pow(img[j].Re, 2) +
            Math.Pow(img[j].Im, 2),0.5);
            dataAbs = Math.Pow(Math.Pow(databaseImg[i, j].Re, 2)
            + Math.Pow(databaseImg[i, j].Im, 2), 0.5);
            temp += Math.Pow(imgAbs - dataAbs,2);
            }
            totalData[i] = Math.Pow(temp,0.5);
      }
      pemenang = findMin(totalData);
      return pemenang;
}
```