BAB IV

IMPLEMENTASI DAN EVALUASI

4.1 Implementasi Sistem

Sebelum dapat mengimplementasikan aplikasi penailian posisi Karate menggunakan sensor Kinect, terdapat beberapa kebutuhan aplikasi yang perlu dipenuhi agar aplikasi ini dapat berjalan dengan semestinya. Kebutuhan yang perlu dipenuhi tersebut dibagi menjadi 2 bagian, yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak. Secara umum kebutuhan perangkat keras yang dicantumkan adalah kebutuhan perangkat keras dalam pengembangan perangkat lunak menggunakan sensor *Kinect* yang tercantum dalam situs web resmi Microsoft.

4.1.1 Kebutuhan Sistem

Dalam proses pengembangan dan implementasi perangkat lunak ini dibutuhkan sumber daya yang terdiri dari perangkat lunak dan perangkat keras. Kebutuhan perangkat lunak mencakup seluruh aplikasi yang diperlukan sebelum aplikasi ini di *deploy*. Sedangkan kebutuhan perangkat keras berupa seluruh alat yang dibutuhkan dalam menjalankan perankat lunak ini dan spesifikasi dari perangkat keras komputer.

a. Kebutuhan Perangkat Keras

Agar dapat menggunakan aplikasi ini secara optimal sebaiknya komputer yang akan mengimplementasikan aplikasi ini memiliki spesifikasi sebagai berikut :

a. Prosesor 32 bit (x86) atau 64 bit (x64) processor

b. Dual-core 2.66-GHz

c. USB 2.0 bus

d. 2 GB RAM

e. Sensor Kinect for Windows atau Sensor Kinect For Xbox 360

Sensor Kinect for Xbox 360 hanya dapat dipakai dalam lingkungan pengembangan aplikasi, sedangkan *Sensor Kinect for Windows* dapat digunakan setelah aplikasi dijalankan di luar lingkungan pengembangan.

Diatas adalah spesifikasi minimum untuk dapat sensor Kinect, namun terdapat beberapa tambahan yaitu :

a. GPU yang mendukung arsitektur DirectX 11

b. Minimal GPU: Nvidia GTX650, AMD Radeon 6950

Tambahan tersebut berasal dari spesifikasi minimum dari program kinect yang dikembangkan menggunakan SDK 1.7 dan 1.8

b. Kebutuhan Perangkat Lunak

Untuk kebutuhan perangkat lunak, aplikasi ini membutuhkan beberapa perangkat lunak yang perlu dipasang agar aplikasi ini dapat berjalan. Kebutuhan perangkat lunak untuk aplikasi ini adalah :

a. Sistem Operasi Windows 7, Windows 8, Windows Embedded Standard 7, atau Windows Embedded POSReady 7

b. .NET Framework 4 atau .NET Framework 4.5

c. Microsoft Speech Platform Runtime (Version 11)

d. Microsoft SQL Server 2008

Sedangkan dalam pengembangan aplikasi ini dibutuhkan beberapa perangkat lunak antara lain :

a. Microsoft Visual Studio 2012
b. Microsoft SQL Server 2008
c. Windows SDK for Kinect 1.7
d. Rational Rose
e. Microsoft Visual Studio
f. Kinect Developer Studio

Beberapa aplikasi tersebut digunakan dalam lingkungan perencanaan dan pendokumentasian aplikasi.

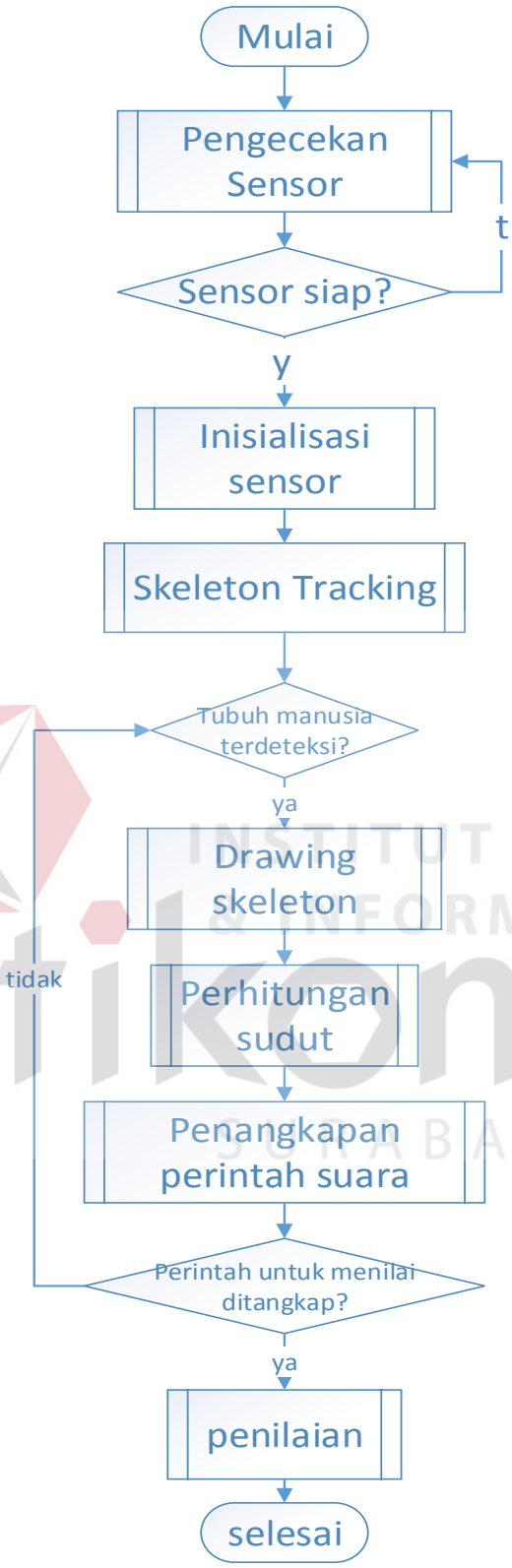
4.2 Proses Implementasi

Aplikasi Penilaian Posisi Karate menggunakan Sensor Kinect ini dibuat berdasarkan rancangan sistem yang telah didesain di tahap sebelumnya. Proses-proses yang dilakukan dalam perangkat lunak ini telah dijelaskan sebelumnya dalam bagian perancangan yang kemudian akan dijadikan acuan dalam proses pengembangan perangkat lunak. Proses-proses tersebut mencakup beberapa modul antara lain pengecekan status sensor, inisialisasi sensor , *skeleton tracking, drawing skeleton,* penangkapan perintah suara dan penilaian.

Langkah awal yang dilakukan dalam membangun aplikasi ini adalah membangun basis data sesuai dengan rancangan yang ada. Basis data ini akan digunakan untuk menyimpan seluruh data yang akan digunakan dalam aplikasi. Proses pembangunan basis data ini menggunakan aplikasi *Power Designer* dimana rancangan struktur basis data akan diolah sehingga menjadi basis data dengan format mdf yaitu ekstensi basis data milik Microsoft SQL Server. Selanjutnya file mdf yang dihasilkan akan dipasang ke dalam *Relational Database Management System Microsoft SQL Server 2008*.

Proses selanjutnya adalah memasang *SDK Kinect for Windows* versi 1.7 yang dapat diperoleh di situs web *Microsoft.* SDK terdiri dari dua bagian yaitu bagian inti dari SDK yang berisi *driver* dari sensor Kinect, dokumentasi *Application Programming Interface(API)*, dan *device interface.* Selanjutnya terdapat *kinect for Windows Developer Toolkit* yang berisi *kinect developer studio* dan contoh aplikasi dasar. Contoh-contoh aplikasi yang disediakan dalam paket ini sangat berguna dalam proses pengembangan perangkat lunak menggunakan sensor Kinect. Selain SDK *Kinect for Windows*, aplikasi ini memerlukan *Microsoft Speech Platform SDK 11* untuk fitur *speech recognizer.* NET Framewrok 4.0 atau 4.5 diperlukan untuk menjalankan perangkat lunak yang dibangun karena aplikasi yang dibangun menggunakan *windows presentation foundation*. Setelah semua SDK dipasang maka komputer mampu untuk mengenali sensor Kinect dan komputer siap untuk membangun perangkat lunak.

Proses pengembangan perangkat lunak ini memiliki beberapa tahapan proses utama. Proses-proses utama ini perlu dibangun satu per satu sesuai dengan rancangan sistem. Proses pembangunan perangkat lunak diawali dengan membangun proses pengecekan sensor dimana jika ditemukan sensor Kinect maka sensor akan diinisialisasi. Selanjutnya dari sensor yang menyala maka akan menghasilkan data yang digunakan dalam proses penggambaran skeleton. Setelah penggambaran *skeleton* akan dilakukan proses perhitungan sudut. Proses *drawing skeleton* dan perhitungan sudut akan dijalankan terus selama proses penangkapan perintah suara tidak mendapat perintah suara untuk melakukan penilaian. Ketika proses penangkapan perintah suara menangkap perintah suara maka akan masuk ke proses penilaian. Alur dari seluruh sub proses ini dapat dilihat Gambar 4.1.



Gambar 4.1 Alur Proses Utama

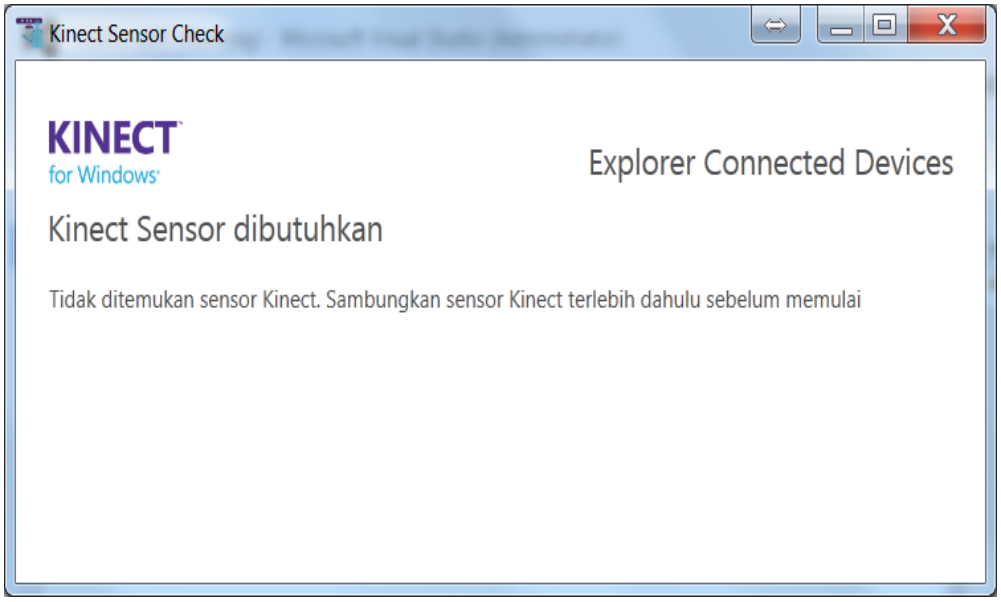
Beberapa bagian utama dari aplikasi ini terdapat dalam beberapa kegiatan utama yang dijelaskan di bawah ini.

4.2.1 Fungsionalitas *Compare*

Sebelum masuk ke fungsionalitas ini, ada beberapa *form* yang akan ditampilkan. Form yang pertama adalah *form* splash screen. Form ini adalah form yang akan muncul saat pertama kali program dijalankan. Dalam form ini terdapat sebuah informasi yang memberitahukan kepada pengguna aplikasi bahwa dibutuhkan sensor Kinect untuk menjalankan aplikasi ini. Selanjutnya pengguna aplikasi dapat menekan tombol masuk untuk memulai aplikasi. Implementasi dari form ini dapat dilihat pada Gambar 4.2.

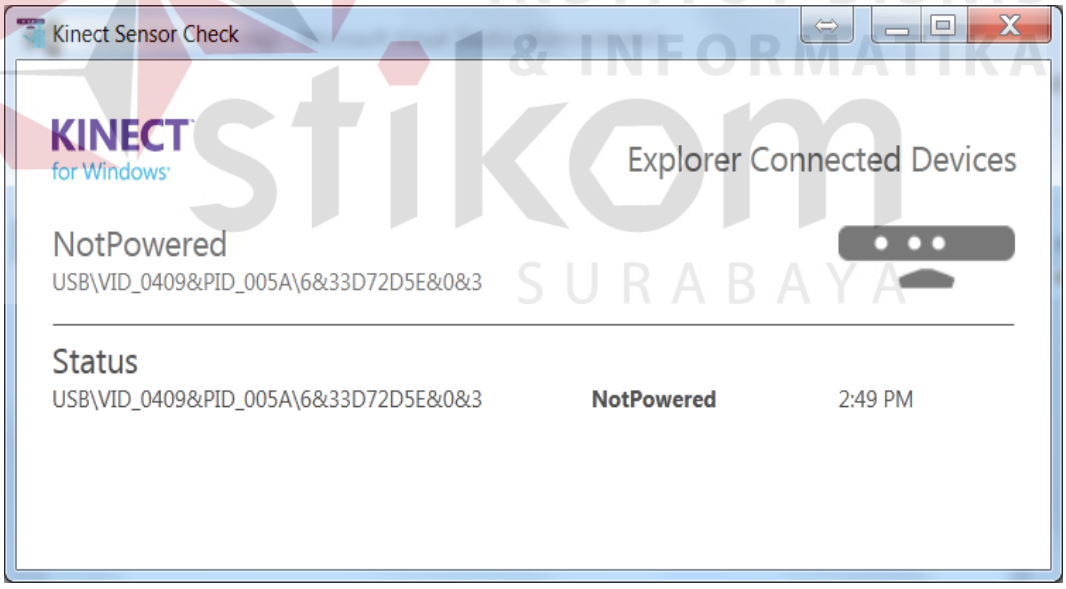
Gambar 4.2 *Splash Screen*

Setelah pengguna menekan tombol masuk, dilakukan pengecekan apakah sudah ada sensor Kinect yang terhubung dengan komputer. Jika tidak ditemukan sensor Kinect, maka aplikasi akan memberitahukan kepada pengguna bahwa tidak ditemukan sensor Kinect seperti tampak pada Gambar 4.3.



Gambar 4.3 *Kinect Sensor Check* dengan Status Tidak Ditemukan Sensor

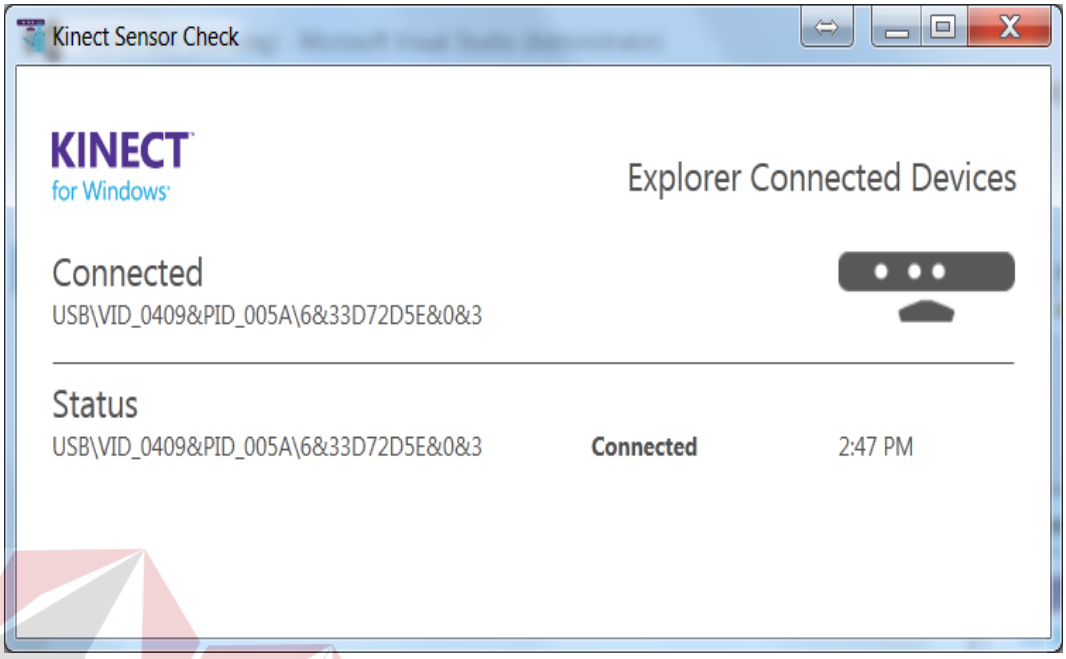
Jika ditemukan sensor Kinect namun sensor tersebut tidak dihubungkan pada catu daya, maka akan muncul peringatan seperti yang tampak pada Gambar 4.4.



Gambar 4.4 *Kinect Sensor Check* dengan Status *Not Powered*

Apabila ditemukan sensor Kinect yang sudah tersambung dengan komputer melalui port USB dan sensor Kinect tersebut sudah mendapat daya listrik, maka

akan muncul informasi seperti pada Gambar 4.5.



Gambar 4.5 *Kinect Sensor Check* dengan Status *Connected*

Selanjutnya setelah berhasil inisialisasi, notifikasi ini akan tetap berjalan selama program menyala untuk mengantisipasi perubahan status dalam aplikasi. Misalnya sensor Kinect terputus di tengah jalannya aplikasi, maka akan muncul status Kinect yang tidak terhubung. Form Menu akan muncul setelah sensor Kinect terhubung.

Proses pengecekan status sensor Kinect ini menggunakan *class KinectSensorItemCollection* dari SDK yang dapat mengecek status sensor mulai dari *connected, disconnected,* dan *not powered.* Ketika ditemukan sensor yang dapat digunakan maka sistem akan menyalakan sensor. Proses pengecekan status sensor ini tampak pada *flowchart* pada Gambar 3.4. Sedangkan untuk proses inisialisasi sensor dapat dilihat pada Gambar 3.5.

Proses pengecekan sensor ini melalui tahap enumerasi dari sensor kinect yang terpasang ke komputer dengan menggunakan kelas KinectSensor dimana

properti dari kelas ini akan berisi dari kumpulan sensor yang terpasang. Penggunaan dari kelas ini tampak pada cuplikan *source code* berikut.

```
private KinectSensor sensor;
foreach (var potentialSensor in KinectSensor.KinectSensors)
{
  if (potentialSensor.Status == KinectStatus.Connected)
  {
    this.sensor = potentialSensor;
    break;
  }
}
```

Dalam proses perulangan diatas, blok program ini akan mengaktifkan sensor Kinect yang memiliki status *connected.* Namun ketika tidak ditemukan sensor Kinect, maka aplikasi akan melakukan perulangan terus menerus sampai ditemukan sensor atau program diakhiri. Ketika ditemukan sensor Kinect, akan dilakukan proses inisialisasi sensor sehingga menghasilkan aliran data yang kita butuhkan. Aliran data yang dapat dipakai antara lain *color, depth,* dan *skeleton*. Proses untuk mengaktifkan aliran data yang dibutuhkan ini dapat dilihat pada source code berikut.

```
if (this.sensor != null)
{
this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
this.sensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
this.sensor.SkeletonStream.Enable();
this.sensor.ColorStream.Enable(ColorImageFormat.InfraredResolution640x480Fps30);
}
```

Setelah aliran data yang dibutuhkan dinyalakan aplikasi akan menyalakan sensor dan sensor akan memberikan hasil pembacaan sensor sesuai dengan aliran data yang dibutuhkan dengan menggunakan perintah `this.sensor.Start()`.

Dari proses pengecekan status ini terdapat beberapa kemungkinan status yang bisa dideteksi. Daftar status sensor kinect yang dapat dideteksi ada dalam Tabel 4.1.

Tabel 4.1 Daftar Status Sensor Kinect

No.	Status	Keterangan
1.	Connected	Sensor Kinect terkoneksi dan siap
2.	DeviceNotGenuine	Sensor Kinect tidak asli
3.	DeviceNotSupported	Sensor kinect tidak mendukung
4.	Disconnected	Sensor kinect tidak terhubung ke *port* USB
5.	Error	Kesalahan terjadi
6.	Initializing	Sensor Kinect dalam proses inisialisasi
7.	InsufficientBandwith	*Port* USB tidak memiliki bandwith yang cukup
8.	NotPowered	Sensor Kinect tidak terhubung dengan catu daya
9.	NotReady	Beberapa bagian sensor belum siap
10.	Undefined	Status tidak terdefinisikan

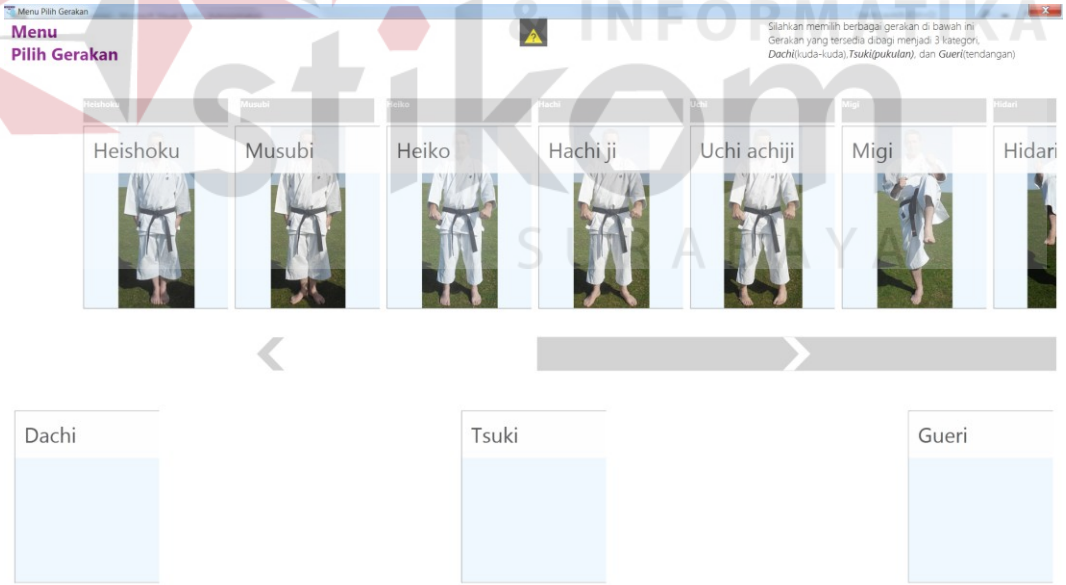
Setelah proses pengecekan dilakukan maka akan dimunculkan tampilan menu utama yang berisi pilihan gerakan. Dimulai dari Form Menu ini dibutuhkan sensor Kinect terhubung dengan komputer namun perangkat lunak masih dapat dikendalikan menggunakan tetikus. Dalam Form Menu ini, sensor Kinect akan menangkap pergerakan telapak tangan pengguna dan menampilkannya pada aplikasi berupa telapak tangan yang berfungsi sebagai tetikus seperti pada Gambar 4.6. Proses inisialisasi *kinect cursor* ini menggunakan *class KinectCursor* dari *namespace Microsoft.Kinect.Toolkit.Controls* yang sudah direferensikan sebelumnya.

Gambar 4.6 Tetikus Virtual

Jika pengguna aplikasi ingin memilih sebuah tombol di dalam aplikasi, pengguna cukup mendekatkan telapak tangannya mendekat ke arah sensor Kinect sampai berubah bentuk seperti pada Gambar 4.7.

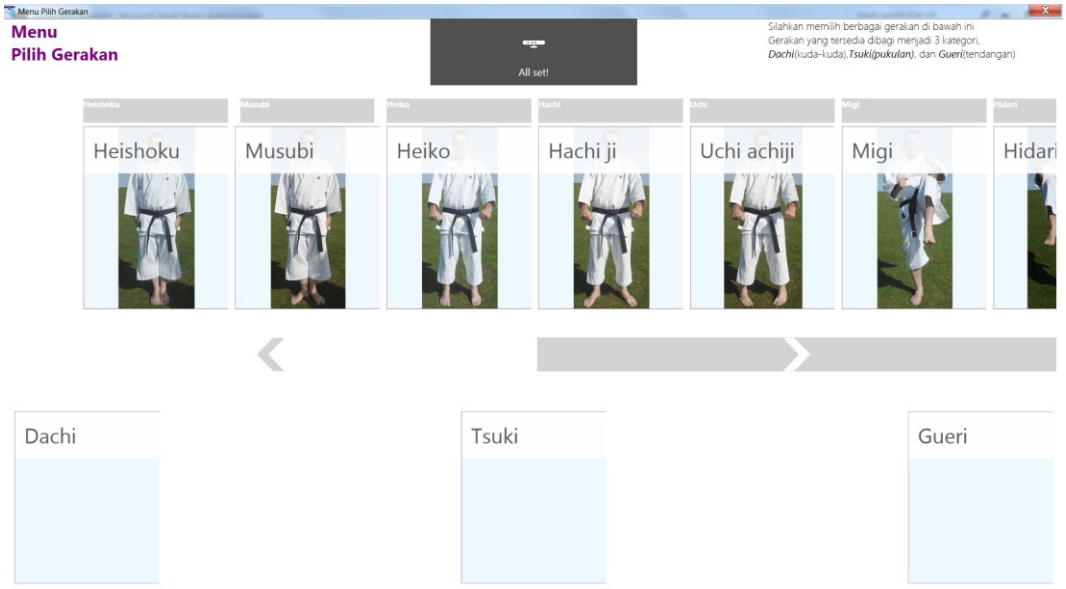
Gambar 4.7 Tetikus Virtual dalam Keadaan didorong

Pada form Menu akan ditampilkan pilihan gerakan yang disajikan berderet secara vertikal dengan tiga tombol sebagai tombol cepat untuk meloncak ke pilihan dengan kategori tertentu. Jika tidak ditemukan sensor Kinect, *user* dapat menggunakan tetikus umum yang biasa digunakan. Tampilan menu jika tidak ditemukan sensor Kinect tampak pada Gambar 4.8.



Gambar 4.8 Form Menu Tanpa Sensor Kinect Terhubung

Jika ditemukan sensor Kinect, makan form menu akan menampilkan pesan notifikasi yang dapat dilihat pada Gambar 4.9.

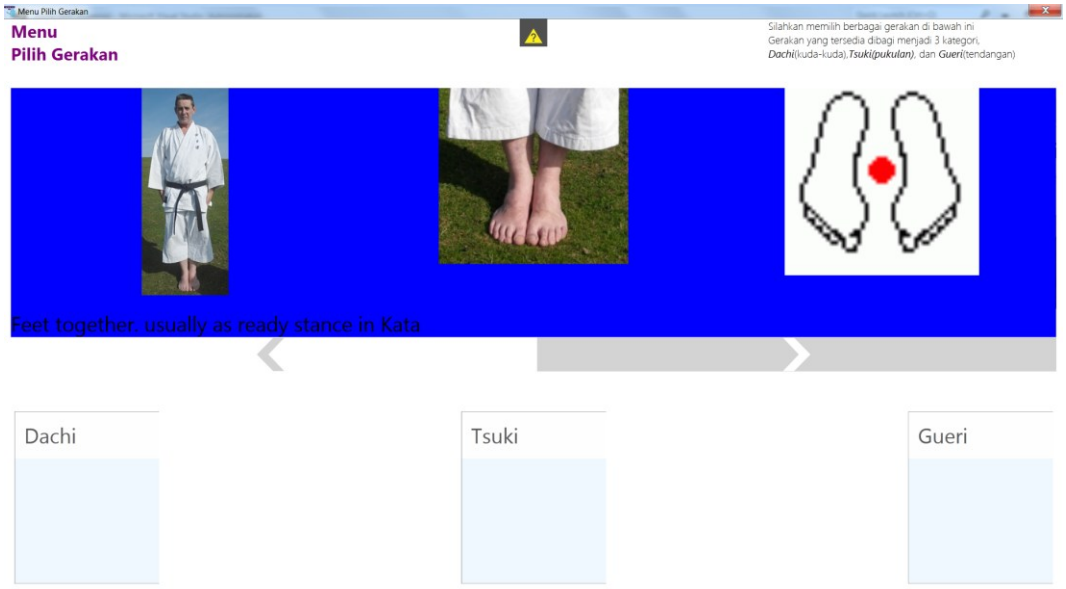


Gambar 4.9 Form Menu dengan Sensor Kinect terhubung

Selanjutnya jika pengguna meletakkan *pointer* di atas nama gerakan diatas *kinect tile button* maka akan ditampilkan Form *Selection Display*. Dalam form ini akan ditampilkan gambar yang mewakili posisi terakhir dari gerakan yang dipilih beserta keterangannya. Jika detail gerakan memiliki lebih dari 1 gambar, maka akan ditampilkan di dalam form ini juga. Sebagai contoh tampak pada Gambar 4.10, gambar di sebelah kiri mewakili posisi tubuh secara keseluruhan, sedangkan pada gambar ditengah fokus pada kaki dimana posisi kaki di gerakan yang bersangkutan merupakan inti posisi tersebut. Gambar selanjutnya merupakan ilustrasi dari telapak kaki yang baik dari posisi tersebut. Seluruh gambar yang terdapat dalam program ini disimpan dalam basis data dalam format tipe data *image*. Proses pengambilan gambar yang ada menggunakan SQL *query* yang kemudian ditampung dalam *bitmap image* yang akan ditampung sementara melalui *memory stream*. Lalu properti *source* dari *imagebox* tempat gambar akan digunakan diisi dengan data

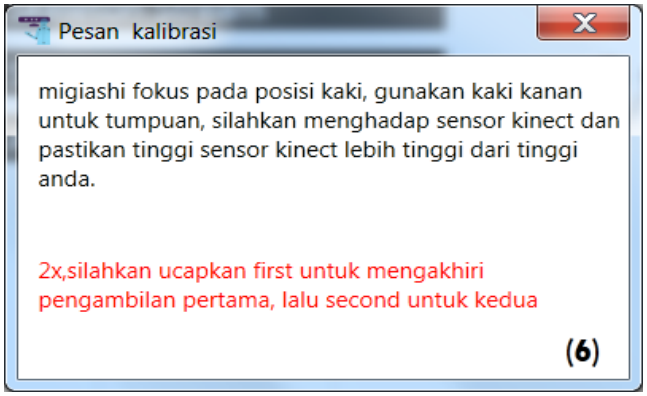
bitmap image yang ada. Sebagai contoh berikut cuplikan *source code* dari fungsi *getImage* dalam form SelectionDisplay.

```
private void getimage()
{
      myConnection.Open();
      DataSet ds = new DataSet();
      SqlDataAdapter sqa = new SqlDataAdapter(
"Select gambarutuh,gambar2,gambar3, keterangan from Informasi_Gerakan where namagerakan
like '"+messageTextBlock.Text+"'", myConnection);
       sqa.Fill(ds);
       myConnection.Close();

       byte[] data1 = (byte[])ds.Tables[0].Rows[0][0];
       MemoryStream strm1 = new MemoryStream();
       strm1.Write(data1, 0, data1.Length);
       strm1.Position = 0;
       System.Drawing.Image img1 = System.Drawing.Image.FromStream(strm1);
       BitmapImage bi1 = new BitmapImage();
       bi1.BeginInit();
       MemoryStream ms1 = new MemoryStream();
       img1.Save(ms1, System.Drawing.Imaging.ImageFormat.Bmp);
       ms1.Seek(0, SeekOrigin.Begin);
       bi1.StreamSource = ms1;
       bi1.EndInit();
       imagebox1.Source = bi1;

       byte[] data2 = (byte[])ds.Tables[0].Rows[0][1];
       MemoryStream strm2 = new MemoryStream();
       strm2.Write(data2, 0, data2.Length);
       strm2.Position = 0;
       System.Drawing.Image img2 = System.Drawing.Image.FromStream(strm2);
       BitmapImage bi2 = new BitmapImage();
       bi2.BeginInit();
       MemoryStream ms2 = new MemoryStream();
       img2.Save(ms2, System.Drawing.Imaging.ImageFormat.Bmp);
       ms2.Seek(0, SeekOrigin.Begin);
       bi2.StreamSource = ms2;
       bi2.EndInit();
       imagebox2.Source = bi2;

       byte[] data3 = (byte[])ds.Tables[0].Rows[0][2];
       MemoryStream strm3 = new MemoryStream();
       strm3.Write(data3, 0, data3.Length);
       strm3.Position = 0;
       System.Drawing.Image img3 = System.Drawing.Image.FromStream(strm3);
       BitmapImage bi3 = new BitmapImage();
       bi3.BeginInit();
       MemoryStream ms3 = new MemoryStream();
       img3.Save(ms3, System.Drawing.Imaging.ImageFormat.Bmp);
       ms3.Seek(0, SeekOrigin.Begin);
       bi3.StreamSource = ms3;
       bi3.EndInit();
       imagebox3.Source = bi3;
}
```



Gambar 4.10 Form Informasi Detail Posisi

Pengguna aplikasi dapat memilih salah satu gerakan dengan cara mendorong tetikus *virtual* yang disediakan diatas pilihan yang diinginkan. Sebelum memuat form *Kinect Window*, sistem akan mengeluarkan pesan notifikasi ke *user* yang berisi informasi bagaimana melakukan proses penilaian. Beberapa hal yang perlu diketahui dalam melakukan proses penilaian ini adalah jumlah pengambilan data dan penekanan penilaian. Contoh pesan notifikasi ini dapat dilihat di Gambar 4.11.



Gambar 4.11 Pesan Notifikasi untuk *Migiashi*

Selanjutnya aplikasi akan menampilkan form *Kinect Window*. Pada form ini terdapat 3 bagian utama, yaitu hasil video yang didapat menggunakan sensor Kinect, pengaturan sensor, dan ilustrasi derajat per join. Pengguna dapat menggunakan perintah suara dalam form ini. Perintah suara yang dapat digunakan dapat dilihat di Tabel 4.2.

Tabel 4.2 Daftar Perintah Suara

No.	Perintah Suara	Fungsi
1.	*Finish*	Perintah suara untuk menandakan *user* telah selesai melakukan gerakan. Hanya berlaku pada gerakan yang membutuhkan 1 kali pengambilan data.
2.	*First*	Perintah suara untuk menandakan *user* telah selesai melakukan gerakan untuk pengambilan data bagian atas tubuh. Hanya berlaku pada gerakan yang membutuhkan 2 kali pengambilan data.
3.	*Second*	Perintah suara untuk menandakan *user* telah selesai melakukan gerakan untuk pengambilan data bagian atas tubuh. Hanya berlaku pada gerakan yang membutuhkan 2 kali pengambilan data.
4.	*Combine*	Perintah suara untuk menggabungkan hasil pengambilan data pertama dan kedua
5.	*Move up*	Menambah sudut elevasi sensor sebanyak 3 derajat
6.	*Move down*	Mengurangi sudut elevasi sensor sebanyak 3 derajat
5.	*Exit*	Keluar dari form penilaian

Proses penyampaian perintah suara dapat dilakukan dengan bantuan dari *class SpeechRecognitionEngine* dari *namespace* Microsoft.Speech.Recognition yang memungkinkan aplikasi mampu untuk mendengarkan perintah suara dari pengguna. *Speech engine* ini akan mendengarkan suara yang ditangkap oleh sensor lalu akan mencocokannya dengan *speech grammar* yang disimpan dalam file XML sesuai dengan perintah suara yang ada dalam Tabel 4.2. Untuk lebih jelasnya proses penangkapan perintah suara dapat dilihat pada alur proses yang dirancang pada Gambar 3.8.

Proses penangkapan perintah suara ini berlangsung selama sensor Kinect menyala. Untuk dapat menggunakan data audio dari sensor dibutuhkan *import* library dari *Microsoft. Speech. AudioFormat* dan *Microsoft. Speech. Recognition*. Dengan kelas dari Mircosoft.Speech ini akan digunakan *Speech Recognition Engine* milik Microsoft. Proses mengaktifkan *speech recognition* ini tampak pada blok program berikut :

```
    RecognizerInfo ri = GetKinectRecognizer();
    if (null != ri)
    {
        this.speechEngine = new SpeechRecognitionEngine(ri.Id);
        using (var memoryStream = new
MemoryStream(Encoding.ASCII.GetBytes(Properties.Resources.SpeechGrammar)))
        {
        var g = new Grammar(memoryStream);
        speechEngine.LoadGrammar(g);
        }
        speechEngine.SpeechRecognized += SpeechRecognized;
        speechEngine.SpeechRecognitionRejected += SpeechRejected;
        speechEngine.SetInputToAudioStream(
        sensor.AudioSource.Start(),
        new SpeechAudioFormatInfo(EncodingFormat.Pcm, 16000, 16, 1, 32000, 2, null));
        speechEngine.RecognizeAsync(RecognizeMode.Multiple);
    }
```

Proses pendeteksian perintah suara ini menggunakan *resources* eksternal berupa file XML yang menampung perintah suara yang ingin digunakan. Masing-masingperintah suara memiliki tag yang unik yang berfunsgi sebagai *identifier*. Tiap identifier dapat menampung beberapa alternatif kata yang seluruh alternatif kata akan men-*trigger event* yang sudah diatur dalam perangkat lunak. Berikut ini adalah cuplikan struktur XML yang berisi tag perintah suara.

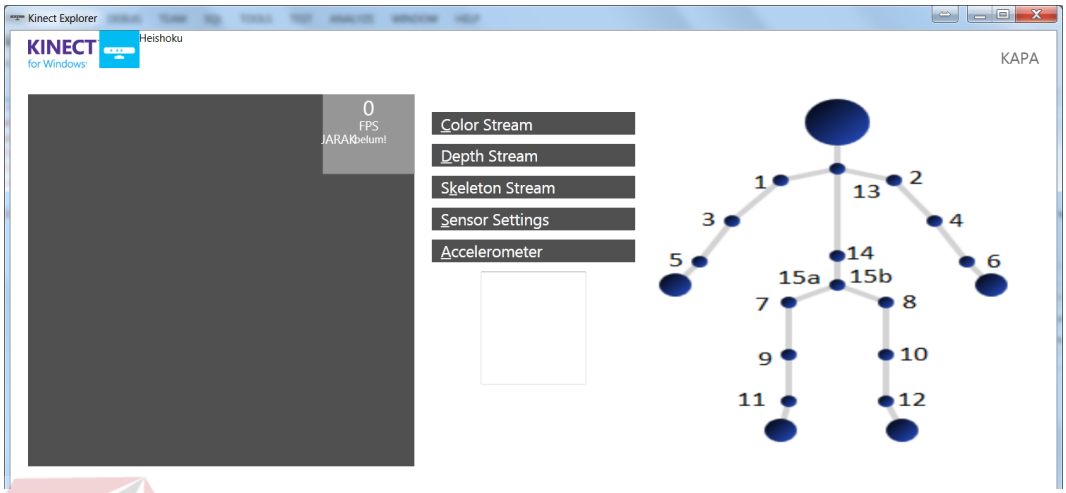
```
<grammar version="1.0" xml:lang="en-US" root="rootRule" tag-format="semantics/1.0-
literals" xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="rootRule">
    <one-of>
      <item>
        <tag>FINISH</tag>
        <one-of>
          <item> finish </item>
          <item> proceed </item>
        </one-of>
      </item>
  </rule>
</grammar>
```

Sensor kinect akan terus memantau data audio yang masuk ke dalam sensor selama sensor menyala. Jika terdapat suara yang didengar, maka perangkat lunak akan menjalankan blok program berikut :

```
private void SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
        {
            const double ConfidenceThreshold = 0.3;
            if (e.Result.Confidence >= ConfidenceThreshold)
            {
                switch (e.Result.Semantics.Value.ToString())
                {
                    case "FINISH":
                        ambillastfull();
                        break;

                    case "EXIT":
                        var childForm = new WindowHasil(0);
                        childForm.Close();
                        break;

                    case "UP":
                        sensor.ElevationAngle = (sensor.ElevationAngle + 3);
                        break;

                    case "DOWN":
                        sensor.ElevationAngle = (sensor.ElevationAngle -17 );
                        break;

                    case "FIRST":
                        ambilatas();
                        MessageBox.Show("atas tersimpan");
                        break;

                    case "SECOND":
                        ambilbawah();
                        MessageBox.Show("bawah tersimpan");
                        break;

                    case "COMBINE":
                        kombinasi();
                        break;
                }
            }
```

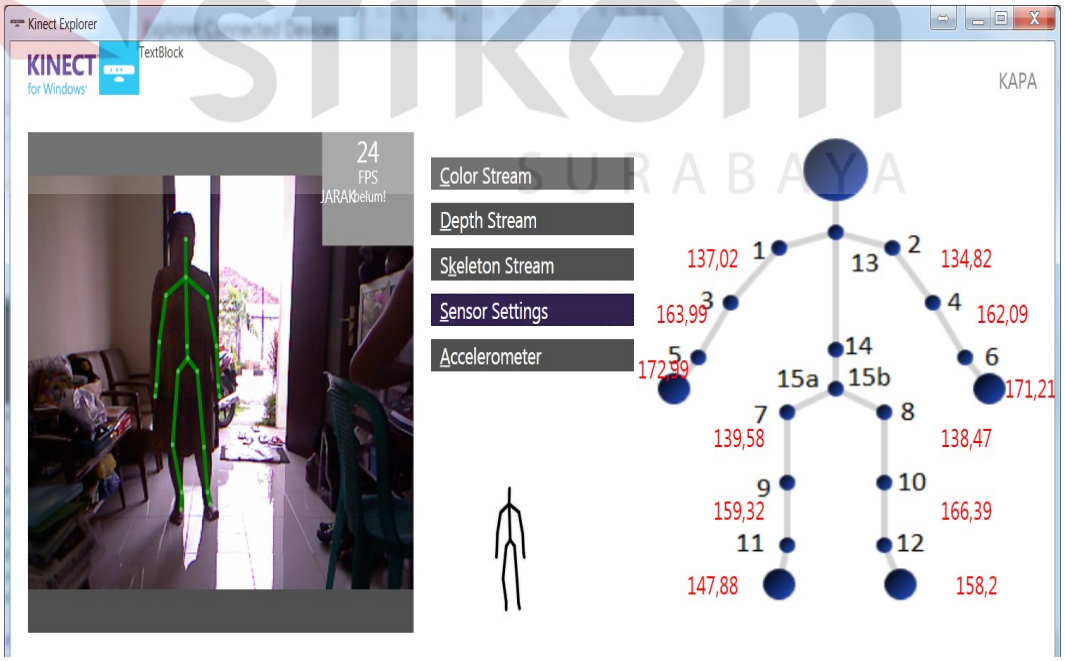
Parameter *Confidence Threshold* merupakan parameter yang menentukan seberapa mirip data suara yang didengar sensor agar dapat terdeteksi dan dianggap sama dengan kamus kata yang ada dalam *speech recognizer engine.* Nilai 1 adalah nilai maksimum dimana data suara harus benar-benar sama dengan data yang ada dalam *speech engine.*

Jika belum ditemukan pengguna di depan sensor Kinect, maka tampilan form ini dapat dilihat pada Gambar 4.12.



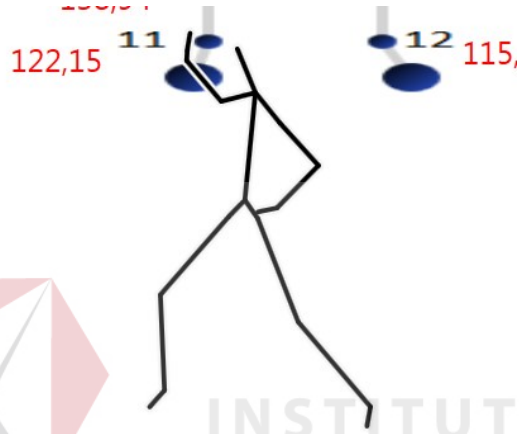
Gambar 4.12 Form *Kinect Window*

Jika ditemukan pengguna di depan sensor *Kinect,* maka form ini akan memproses hasil bacaan sensor dan menampilkannya seperti yang dapat dilihat pada Gambar 4.13.



Gambar 4.13 Form *Kinect Window* dengan Hasil *Skeleton Tracking*

Proses pendeteksian tubuh manusia atau yang disebut *skeleton tracking* ini memiliki alur seperti yang sudah dirancang pada Gambar 3.6. Setelah berhasil di deteksi, perangkat lunak akan memvisualisasikan bentuk *skeleton* yang didapat. Proses *drawing skeleton* ini dijelaskan melalui *flowchart* di Gambar 3.7. Hasil dari proses visualisasi *skeleton* ini berupa *stickman* yang tampak seperti pada Gambar 4.14.



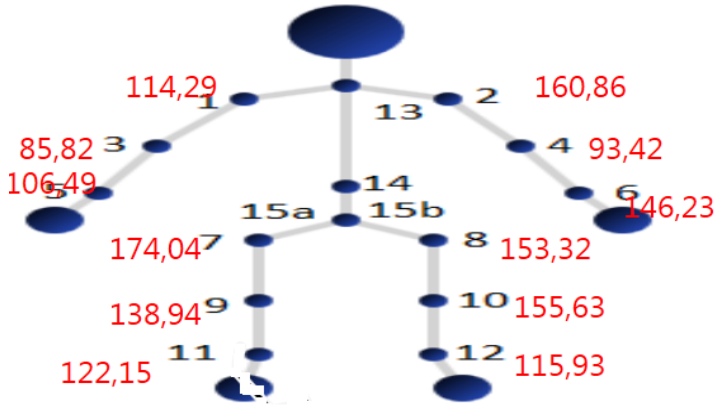
Gambar 4.14 Hasil Visualisasi *Skeleton*

Proses dari penggambaran *stickman* ini menggunakan *method DrawStickman* dimana *method* ini memiliki fungsi untuk mengambil data *skeleton* yang memiliki 3 parameter yaitu data *skeleton*, jenis *brush* yang digunakan, dan ketebalan garis. Data *skeleton* berasal dari *array skeleton yang sudah didefinisikan sebelumnya yaitu* `private Skeleton[] allSkeletons =new Skeleton[skeletonCount].` Ketika method ini dijalankan, *canvas stickman* akan diisi oleh garis-garis yang saling terhubung dari variabel *point* yang terdiri dari koordinat x dan y dari tiap *joint*.

Proses perbandingan data *skeleton* diolah dari data mentah *skeleton* yang didapat dari *skeleton stream*. Proses pemrosesan data mentah dari *skeleton* yang terdeteksi menjadi sudut-sudut yang diperlukan untuk proses penilaian tampak pada Gambar 3.9. Sensor Kinect menghasilkan joint yang memiliki 3 nilai, yaitu X,Y, dan Z. Selain 3 parameter tersebut, dibutuhkan panjang vektor yang didapat dari perhitungan $\sqrt{x^2 + y^2 + z^2}$ sudut dari dua vektor a dan vektor b adalah $cos\theta = a.b \ / \ |a||b|$. Hasil dari a.b dan |a||b| dihitung dalam *method* kelas *Measurement Lib*. Dari metode yang ada di kelas ini akan dihasilkan derajat per join yang diproses secara *realtime* ditampilkan dalam program. Hasil perhitungan sudut dari tubuh pengguna perangkat lunak tampak pada Gambar 4.15.

Dalam perangkat lunak, proses perhitungan sudut ini dibagi menjadi beberapa sub proses antara lain perhitungan |vektor| (panjang vektor) yang didapat dari perhitungan melalui `return (float)Math.Sqrt(X * X + Y * Y + Z * Z).` Lalu juga dihitung nilai perkalian *dot product* dari dua vektor dengan menggunakan `hasilperkaliandot = Vector3.Dot(vektorA, vektorB);` Selanjutnya dari perhitungan tersebut masing masing vektor akan dihitung nilai derajatnya melalui

```
return (float)Math.Round((Math.Acos(hasilperkaliandot) * 180 / Math.PI), 2)
```

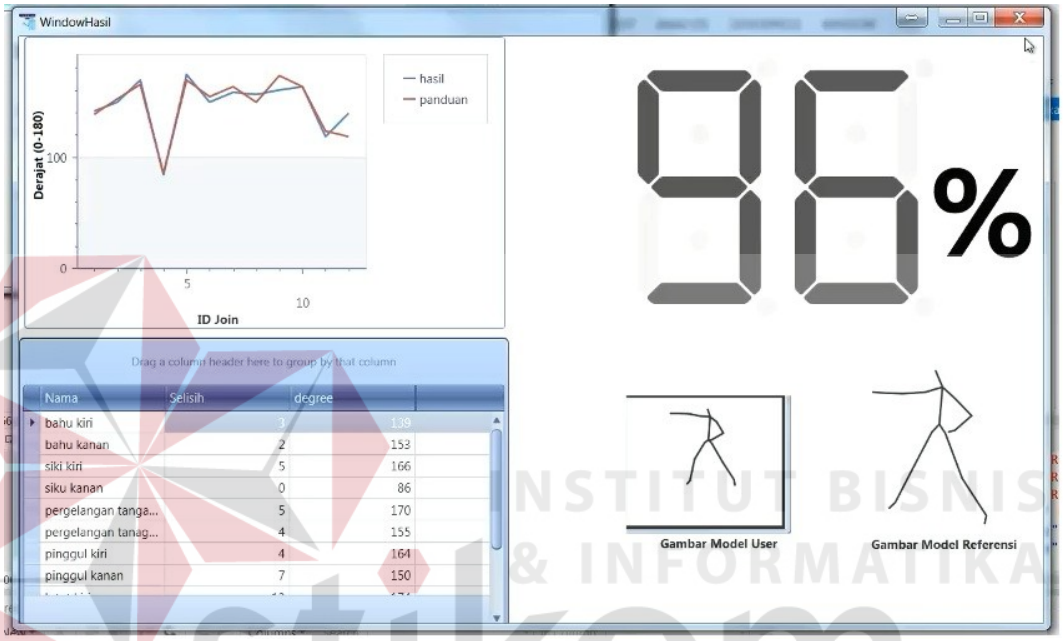


Gambar 4.15 Hasil Perhitungan Sudut per Sendi

Nilai yang didapat akan mengikuti pergerakan skeleton secara *realtime*. Nilai dari sudut-sudut ini akan diambil ketika *event speechRecognized* dengan perintah suara *combine* atau *finish.* Ketika hal ini terjadi, nilai dari seluruh sudut akan diambil dan ditampung ke dalam tabel *charting*. Nilai 12 sudut dari pengguna aplikasi akan dimasukkan bersamaan dengan nilai 12 sudut *role model* yang ada di dalam basis data dan ditampung dalam tabel yang sama dengan menggunakan pembeda berupa nilai IDTIPE. Nilai 'P' merupakan tanda untuk mengidentifikasi bahwa nilai pada baris tersebut adalah nilai sudut dari data *role model* (panduan) sedangkan nilai 'H' menunjukkan bahwa nilai sudut yang ada merupakan hasil dari data pengguna yang sudah diambil. Proses ini membutuhkan 2 parameter antara lain id joint dan id tipe. *Query* dari proses ini adalah "`UPDATE charting SET degree = CASE IDTIPE WHEN 'p' THEN @p WHEN 'h' THEN @l END WHERE idjoint IN (@jointid)`". Dari hasil perhitungan sudut yang sudah dilakukan, sudut tersebut dapat disimpan dalam basis data berupa nilai dari 12 *joint* yang digunakan dalam metode setRealisasi dalam kelas *compare.* Selanjutnya dengan menggunakan metode getHasil, akan didapat rata-rata selisih dari ke dua belas sudut yang dibandingkan dengan data referensi yang sudah diambil melalui metode setPanduan. Metode getHasil memberikan nilai return persentase selisih dari *joint* yang didapat dari membandingkan nilai panduan dan nilai hasil melalui yang ditampung dalam variabel hasil yang diproses menggunakan blok program berikut: `hasil=100-(((Math.Abs(panduan - realisasi))/realisasi)*100).` Proses selanjutnya dari data yang didapat akan diproses ke fungsionalitas *feedback*..

4.2.2 Fungsionalitas *Feedback*

Fungsionalitas *feedback* fokus pada penyajian data hasil penilaian. Setelah proses pengambilan data selesai, data akan diolah dan ditampilkan dalam sebuah form yang menyajikan informasi mengenai hasil dari gerakan yang dilakukan. Contoh tampilan form ini tampak pada Gambar 4.16.



Gambar 4.16 Form Hasil Penilaian

Di dalam form ini dibagi menjadi 4 bagian, yaitu grafik perbandingan antara data panduan dan data realisasi, nilai hasil akhir, *data grid view* yang berisi data mentah, dan visualisasi perbandingan *skeleton*. Dalam proses sebelumnya diambil seluruh derajat sudut melalui fungsional *save* yang selanjutnya akan menghasilkan nilai dari 12 sudut yang merupakan konfigurasi sudut dari tiap sendi yang digunakan. Nilai-nilai dari sudut ini lalu dibandingkan selisihnya dengan data referensi yang ada lalu selisih ke dua belas joint ditambah dan hasilnya dibagi 12 sehingga didapat rata-rata selisih sudut dari posisi yang diproses dengan data

referensi. Proses penilaian ini melalui alur proses seperti pada *flowchart* di Gambar 3.10.

Perhitungan nilai rata-rata dalam perangkat lunak membutuhkan data dari panduan yang sesuai dengan id gerakan yang dicoba dan data aktual dari pengguna perangkat lunak. Variabel pl1,pl3,pl5,pl7,pl9 dan pl11 adalah nilai dari sudut data panduan tubuh bagian kiri dan pr2,pr4,pr6,pr8,pr10,pr12 adalah nilai dari sudut data panduan tubuh bagian kanan. Variabel ll1,ll3,ll5,ll7,ll9 dan ll11 adalah nilai dari sudut data tubuh *user* bagian kiri dan lr2,lr4,lr6,lr8,lr10,lr12 adalah nilai dari sudut data panduan tubuh *user* bagian kanan. Nilai rata-rata yang didapat ditampung dalam variabel *average* yang nilainya diambil dari rata-rata nilai array dengan nama arraynilai. Proses perhitungan ini menggunakan metode getHasil dalam kelas *compare* dimana method ini melakukan kegiatan perhitungan sebagai berikut :

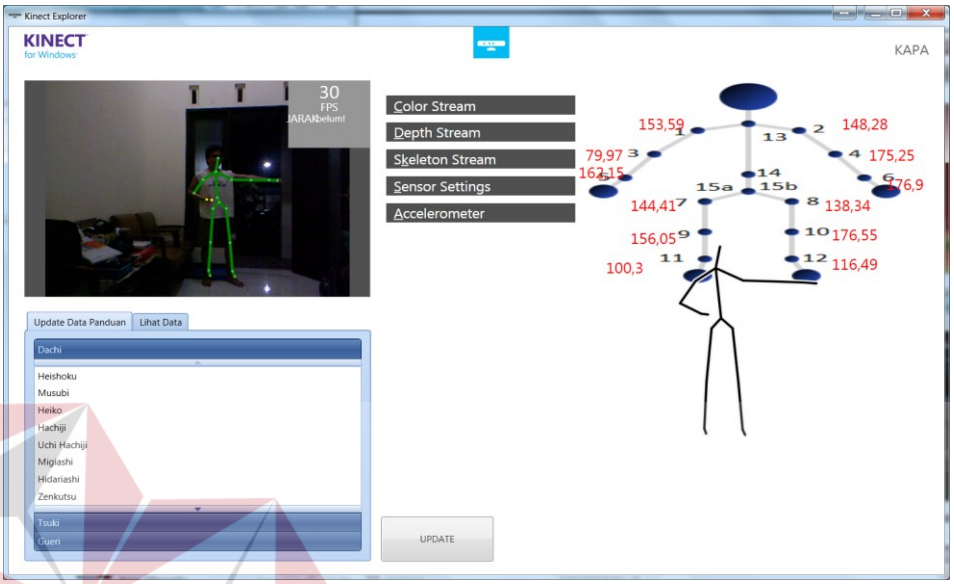
```
double[] arraynilai = {
                      (100 - ((Math.Abs(pl1 - ll1) / pl1) * 100)),
                      (100 - ((Math.Abs(pl3 - ll3) / pl3) * 100)),
                      (100 - ((Math.Abs(pl5 - ll5) / pl5) * 100)),
                      (100 - ((Math.Abs(pl7 - ll7) / pl7) * 100)),
                      (100 - ((Math.Abs(pl9 - ll9) / pl9) * 100)),
                      (100 - ((Math.Abs(pl11 - ll11) / pl11) * 100)),
                      (100 - ((Math.Abs(pr2 - lr2) / pr2) * 100)),
                      (100 - ((Math.Abs(pr4 - lr4) / pr4) * 100)),
                      (100 - ((Math.Abs(pr6 - lr6) / pr6) * 100)),
                      (100 - ((Math.Abs(pr8 - lr8) / pr8) * 100)),
                      (100 - ((Math.Abs(pr10 - lr10) / pr10) * 100)),
                      (100 - ((Math.Abs(pr12 - lr12) / pr12) * 100)),
                      };
int average = Convert.ToInt32(arraynilai.Average());
```

Nilai average ini merupakan parameter ketika Window Hasil diaktifkan dimana nilai ini akan tampil pada kolom kanan atas Window Hasil.

4.2.3 Fungsionalitas *Save*

Fungsionalitas *save* fokus pada pengambilan data persendian dari model yang akan dijadikan acuan dalam aplikasi. Fungsionalitas ini bersifat *stand alone* dan tidak terdapat pada program utama dimana fungsionalitas ini ditujukan pada *admin*. Fungsionalitas ini akan digunakan oleh *admin* untuk memasukkan atau

mengubah data panduan yang ada dalam basis data. Fungsionalitas ini menggunakan form *update* yang dikembangkan dari form proses penilaian gerakan. Tampilan dari form *update* dapat dilihat di Gambar 4.17.



Gambar 4.17 Form *Update* Data Panduan

Untuk mengubah data yang ada di basis data, *admin* memilih gerakan yang ingin diubah pada pilihan gerakan dan selanjutnya model dapat melakukan gerakan yang diinginkan lalu *admin* memilih tombol *update*. Form ini juga dilengkapi dengan fungsi untuk melihat data panduan dari tiap gerakan yang disediakan oleh aplikasi. Orang yang akan diambil posisi tubuhnya dapat mempraktekkan gerakan yang akan disimpan lalu mempertahankan posisi akhir dari gerakan tersebut lalu admin akan memilih nama gerakan yang akan disimpan dengan data referensi yang didapat dari posisi tubuh orang tersebut dan mengakhiri proses perekaman data dengan menekan tombol *update*. Proses penggambaran *stickman* dan perhitungan sudut melalui proses yang sama seperti yang sudah dijelaskan dalam fungsionalitas *compare*. Yang membedakan fungsionalitas compare dan save adalah aliran data ke basis data pada tabel panduan. Fungsionalitas *save* hanya memiliki proses untuk

menulis ke basis data dengan tujuan untuk memperbarui data panduan. Sedangkan pada fungsionalitas compare data panduan hanya diambil untuk digunakan dalam proses penilaian tanpa melakukan proses *write*.

Proses perekaman model sebagai data referensi melalui proses yang sama seperti fungsionalitas *compare* dimana perbedaan terletak pada proses perekaman model hanya menggunakan metode setRealisasi dan selanjutnya akan disimpan nilai selisih sudut ke 12 joint ke dalam basis data. Proses perekaman model sebagai data referensi yang dilakukan perangkat lunak ini melalui proses-proses yang sesuai dengan *flowchart* pada Gambar 3.11. Selain data dari 12 sudut *skeleton* yang disimpan, perangkat lunak juga mengambil gambar *stickman* dan menyimpannya ke dalam basis data. Proses pengambilan data ini dibagi menjadi 2, yaitu pengambilan data secara keseluruhan dan parsial. Pengambilan data secara keseluruhan dilakukan ketika posisi tubuh memungkinkan untuk dibandingkan dari satu perspektif. Sedangkan pengambilan parsial dilakukan dua kali dimana pengambilan pertama mengambil nilai *joint* bagian tubuh atas (l1,r2,l3,r4,l5,r6) lalu pengambilan kedua mengambil nilai *joint* bagian tubuh bawah (l7,r8,l9,r10,l11,r12). Query yang diperlukan dalam melakukan pengambilan data secara keseluruhan adalah `"update panduan set l1=@l1,r2=@r2,l3=@l3,r4=@r4,l5=@l5,r6=@r6,l7=@l7,r8=@r8,l9=@l9,r10=@r10,l11=@l11,r12=@r12 where idgerakan="` `+ idgerakan.` Sedangkan proses pengambilan parsial menggunakan parameter update yang sesuai.

Selain data dari *joint,* fungsionalitas *save* juga melakukan kegiatan penyimpanan gambar hasil visualisasi *stickman* ke dalam basis data. Proses penyimpanan ini dilakukan dengan cara melakukan proses *screen shot* area *canvas*

stickman dan menyimpannya ke dalam file dengan ekstensi png pada lokasi statis yang sudah ditentukan. Proses ini dilakukan melalui blok program berikut.

```
public void SaveScreen(double x, double y, double width, double height)
{
    int ix, iy, iw, ih;
    ix = Convert.ToInt32(x);
    iy = Convert.ToInt32(y);
    iw = Convert.ToInt32(width);
    ih = Convert.ToInt32(height);
    try{
          System.Drawing.Bitmap myImage = new System.Drawing.Bitmap(iw, ih);
          System.Drawing.Graphics gr1 = System.Drawing.Graphics.FromImage(myImage);
          IntPtr dc1 = gr1.GetHdc();
          IntPtr dc2=NativeMethods.GetWindowDC(NativeMethods.GetForegroundWindow());
          NativeMethods.BitBlt(dc1, ix, iy, iw, ih, dc2, ix, iy, 13369376);
          gr1.ReleaseHdc(dc1);
          myImage.Save(path, System.Drawing.Imaging.ImageFormat.Png);
        }
    catch { }
}
```

Parameter yang diperlukan adalah koordinat x dan y yang berfungsi untuk menyimpan lokasi *top* dan *left* dari *canvas stickman.* Parameter lainnya adalah panjang dan lebar dari *canvas* tersebut. Untuk dapat menjalankan proses diatas, aplikasi perlu mengimpor data library yang diperlukan. Untuk dapat mengimpor library tersebut aplikasi memerlukan *namespace* dari sistem operasi yaitu `System.Runtime.InteropServices`. Setelah namespace terdaftar, program melalui blok program berikut akan mengimpor library yang diperlukan untuk menjalankan method *SaveScreen.*

```
internal class NativeMethods
{
    [DllImport("user32.dll")]
    public extern static IntPtr GetDesktopWindow();
    [DllImport("user32.dll")]
    public static extern IntPtr GetWindowDC(IntPtr hwnd);
    [DllImport("user32.dll", CharSet = CharSet.Auto, ExactSpelling = true)]
    public static extern IntPtr GetForegroundWindow();
    [DllImport("gdi32.dll")]
    public static extern UInt64 BitBlt(IntPtr hDestDC, int x, int y, int nWidth, int
nHeight, IntPtr hSrcDC, int xSrc, int ySrc, System.Int32 dwRop);
}
```

Setelah image disimpan sesuai dengan *path* yang ditentukan, perangkat lunak akan memuat gambar tersebut dan menampungnya dengan tipe data *image.* Proses ini melibatkan *binary large object file* (BLOB) dimana akan diproses dengan menggunakan kelas *FileStream* dari *namespace Sysyem.IO.* Proses penyimpanan

gambar panduan berupa *stickman* ke dalam basis data ini dilakukan melalui *method berikut.*

```
public void insertgambarpanduan()
{
     SqlCommand cmd = new SqlCommand(
     "update PERBANDINGANGAMBAR set GAMBARPANDUAN=@BLOBData1 WHERE IDGERAKAN=" +
     FileStream fsBLOBFile1 = new FileStream(path, FileMode.Open, FileAccess.Read);
     Byte[] bytBLOBData1 = new Byte[fsBLOBFile1.Length];
     fsBLOBFile1.Read(bytBLOBData1, 0, bytBLOBData1.Length);
     fsBLOBFile1.Close();
     SqlParameter prm1 = new SqlParameter("@BLOBData1", SqlDbType.VarBinary,
     bytBLOBData1.Length, ParameterDirection.Input, false,
     0, 0, null, DataRowVersion.Current, bytBLOBData1);

     cmd.Parameters.Add(prm1);

     myConnection.Open();

     cmd.ExecuteNonQuery();
     System.Windows.MessageBox.Show("Picture has been uploaded");
     myConnection.Close();
}
```

4.3 Evaluasi Sistem

Sistem akan dievaluasi melalui uji coba dengan metode *black box* dan akan dilakukan uji coba kepada orang yang memiliki pengetahuan tentang *Karate* yang berbeda.

4.3.1 Hasil Uji Coba dengan Metode Black Box

Berikut ini adalah laporan tes dari hasil uji coba aplikasi. Hasil uji coba aplikasi ini dilengkapi dengan *screen shot* hasil uji coba yang ada di Lampiran 5.

a. Fungsionalitas memilih gerakan

   Hasil *test report* dari fungsionalitas ini dapat dilihat di Tabel 4.3.

Tabel 4.3 Hasil Uji Coba Fungsionalitas Memilih Gerakan

Flow	**Hasil yang diharapkan**	**Hasil uji coba**	
Main Flow		**Result**	
Aktor	**Sistem**	**kondisi**	**bukti**
Memilih Gerakan dengan menggunakan Sensor Kinect			
1. Menjalankan aplikasi	H1. *Splash screen* ditampilkan	Lolos	a.1
2. Menekan tombol masuk	H2. Menu Pilihan ditampilkan	Lolos	a.2
3. Inisialisasi sensor Kinect	H3. Sensor Kinect menyala & tersambung	Lolos	a.3
4. Menggerakan telapak tangan di depan sensor	H4. *Kinect hand cursor* ditampilkan	Lolos	a.4
5. Menggengam dan menggeser tangan di pilihan "*heiko"*	H5. Pilihan gerakan akan bergeser ke arah tangan bergeser dengan patokan *tile "heiko"*	Lolos	a.5
6. Meletakkan kinect hand cursor diatas informasi detil "*heishoku*"	H6. Selection display dari "*heishoku*"ditampilkan	Lolos	a.6
7. Memilih gerakan "*heishoku*" dengan cara mendorong telapak tangan mendekati sensor	H7. *Kinect Window* dengan parameter *"heishoku"* ditampilkan	Lolos	a.7
Alternate Flow		**kondisi**	**bukti**
Memilih Gerakan tanpa menggunakan Sensor Kinect			
1. Menjalankan aplikasi	H1. *Splash screen* ditampilkan	Lolos	a.8
2. Menekan tombol masuk	H2. Menu Pilihan ditampilkan	Lolos	a.9
3. Inisialisasi sensor Kinect	H3. Tidak ditemukan sensor	Lolos	a.10
4. Meletakkan cursor diatas informasi detil "*heishoku*"	H4. Selection display dari *"heishoku*" ditampilkan	Lolos	a.11
5. Meletakkan cursor diatas tombol panah kanan	H5. Pilihan gerakan akan bergeser ke kanan	Lolos	a.12
6. Memilih gerakan dengan cara mengklik *tile "heishoku"*	H6. Pesan notifikasi dibutuhkan sensor kinect	Lolos	a.13

b. Fungsionalitas penilaian gerakan.

Hasil *test report* dari fungsionalitas ini dapat dilihat di Tabel 4.4.

Tabel 4.4 Hasil Uji Coba Fungsionalitas Menilai Gerakan

Flow	**Hasil yang diharapkan**	**Hasil uji coba**	
Main Flow		***Result***	
Aktor	**Sistem**	**Kondisi**	**bukti**
Menilai Gerakan			
1. Memilih gerakan "heishoku"	H1. *Kinect Window* ditampilkan dengan nama gerakan *"heishoku"*	Lolos	b.1
2. Inisialisasi sensor Kinect	H2. Sensor Kinect menyala	Lolos	b.2
3. Berada di posisi dimana sensor Kinect dapat membaca pergerakan	H3. Skeleton tracking ditampilkan	Lolos	b.3
4. Melakukan gerakan dan pada akhir gerakan mengucapkan "finish"	H4. Window Hasil penilaian *Heishoku"* ditampilkan	Lolos	b.4
Alternate Flow			
Tidak ada			

c. Fungsionalitas memasukkan data

Hasil *test report* dari fungsionalitas ini dapat dilihat di Tabel 4.5.

Tabel 4.5 Hasil Uji Coba Fungsionalitas Pembaruan Data

Flow	**Hasil yang diharapkan**	**Hasil uji coba**	
Main Flow		***Result***	
Aktor	**Sistem**	**Kondisi**	**Bukti**
Update Data Gerakan			
1. Menekan tombol Ctrl+H di saat yang bersamaan	H1. Form Data ditampilkan	Lolos	c.1
2. Memilih gerakan "ujic coba test" yang diupdate	H2. Data *joint* ditampilkan di *data gridview*	Lolos	c.2
3. Sensor Kinect dinisialisasi	H3. Sensor Kinect menyala & tersambung	Lolos	c.3
4. Hasil baca sensor disampaikan ke aplikasi	H4. Data 12 *joint* ditampilkan	Lolos	c.4
5. Admin memilih tombol simpan	H5. Tabel Panduan dengan nama "heishoku" ter-*update*	Lolos	c.5
Alternate Flow		**Kondisi**	**Bukti**
Tidak ada			

4.3.2 Hasil Uji Coba oleh Pengguna

Aplikasi ini telah dicoba oleh penulis dengan menggunakan data model dari karatekawan dengan tingkat sabuk cokelat. Data model terdiri dari 24 posisi yang masing-masing posisi akhir dari berbagai *kihon*. Proses uji coba dilakukan dengan melakukan 24 gerakan sebanyak masing-masing 10 kali. *Kihon* yang dicoba terdiri dari 11 posisi kuda-kuda, 9 posisi pukulan, dan 4 posisi tendangan. Hasil uji coba yang sudah dilakukan tampak pada Tabel 4.6.

Tabel 4.6 Hasil Uji Coba Rata-rata Penilaian

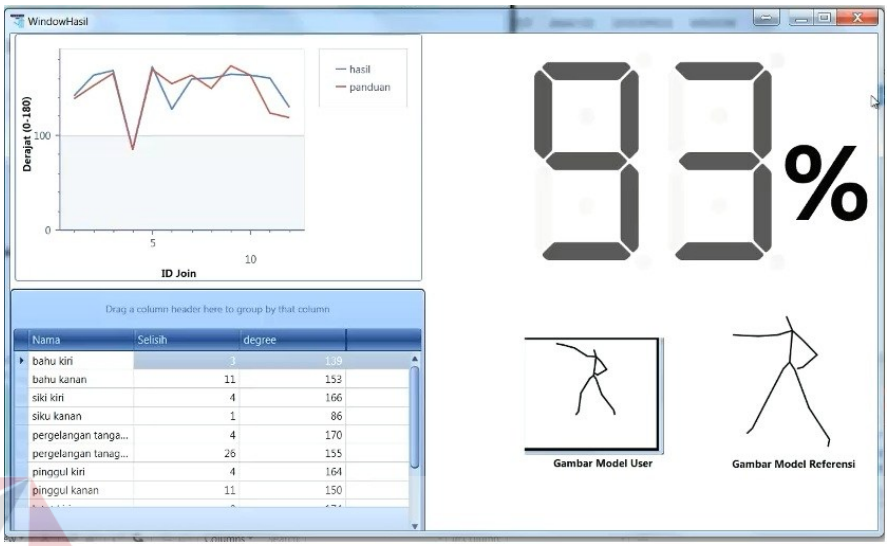
No	Nama Gerakan	Rata-rata penilaian
1	*heishoku dachi*	96,6
2	*musubi dachi*	88,9
3	*heiko dachi*	96,4
4	*hachi-ji dachi*	88,4
5	*uchi-hachiji-dachi*	89,4
6	*migi-ashi dachi*	79,7
7	*hidari-ashi dachi*	71,8
8	*zenkutsu dachi*	84
9	*kokutsu dachi*	86,2
10	*fudo dachi*	88,2
11	*hangetsu dachi*	85,5
12	*oi tsuki*	91,8
13	*gyako tsuki*	79,4
14	*mawashi tsuki*	85
15	*kizami tsuki*	88,2
16	*awase tsuki*	91
17	*hasami tsuki*	90,7
18	*otsohi tsuki*	88,7
19	*yoko tsuki*	90,6
20	*heiko tsuki*	89
21	*maegueri kekomi*	59
22	*yokogueri kekomi*	66,7
23	*yokogueri keage*	57,9
24	*maehiza gueri*	79,4

Dari hasil uji coba, didapatkan rata-rata 83,85% untuk semua gerakan yang dicoba dengan detail 86,8% untuk kuda-kuda, 88,2% untuk pukulan, dan 65% untuk tendangan. Hasil diatas merupakan tingkat keakurasian data dari data model dengan data hasil akuisisi *skeleton* pengguna. Proses uji coba yang dilakukan menggunakan 24 gerakan yang menggunakan data referensi dari role model karatekawan sabuk cokelat. Untuk hasil yang lebih mendetail mengenai hasil uji coba yang dilakukan dapat dilihat pada Lampiran 6.

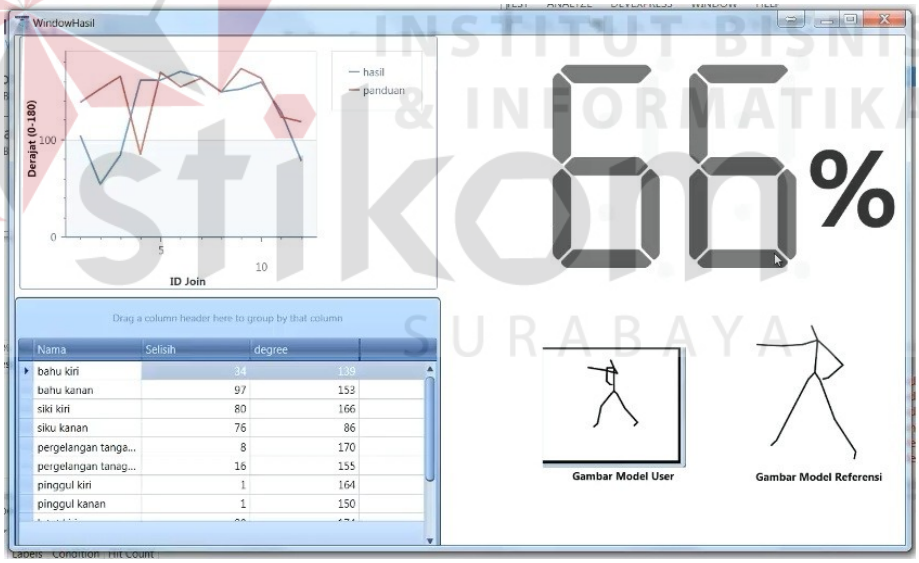
Selain uji coba diatas, dilakukan ujicoba untuk mengetahui kemampuan aplikasi dalam menilai sebuah gerakan yang tingkat kesalahannya bersifat mayor dan minor. Dalam kesalahan yang bersifat mayor, pengguna aplikasi melakukan kesalahan yang benar-benar melenceng dari gerakan yang dipilih. Sedangkan untuk kesalahan minor pengguna aplikasi melakukan gerakan yang berakhir di sebuah posisi dimana posisi ini secara garis besar sesuai dengan data model namun memiliki tingkat perbedaan yang cukup kecil dengan data model. Dalam skenario uji coba pengguna aplikasi memeragakan posisi dengan sebaik-baiknya sesuai dengan petunjuk. Selanjutnya dilakukan penilaian dengan melakukan sedikit penyimpangan dari posisi model. Dua buah skenario ini mewakili kesalahan yang bersifat minor. Sedangkan yang terakhir dilakukan uji coba dengan melakukan gerakan yang sama sekali berbeda untuk mewakili kesalahan mayor.

Untuk uji coba minor *oi tsuki* dipilih untuk diuji. Pengambilan kedua dilakukan dengan menambah sudut elevasi bahu kiri agar arah pukulan menuju ke kepala lawan. Sedangkan pengambilan ketiga menggunakan *gyako tsuki,* dimana gerakan ini sebenarnya sama dengan *oi tsuki* namun menggunakan tangan yang

berbeda. Hasil pengambilan pertama dapat dilihat pada Gambar 4.16 dengan hasil 96%. Untuk pengambilan kedua dengan hasil 93% dapat dilihat di Gambar 4.18.



Gambar 4.18 Uji Coba *oi tsuki* kedua



Gambar 4.19 Uji Coba *Oi tsuki* ketiga

Pada Gambar 4.19 tampak hasil uji coba dengan memeragakan pukulan dengan tangan kanan dimana semestinya yang diperlukan adalah pukulan menggunakan tangan kiri. Hasil uji coba ini memiliki hasil 66% dimana kesalahan terjadi secara signifikan yang diakibatkan dari data bagian tubuh yang tidak sesuai

dengan data model pada bagian tangan. Hasil uji coba pertama memiliki nilai lebih baik karena kesalahan yang dilakukan bersifat minor dan hanya terletak pada bagian tangan kiri yang memiliki nilai sudut elevasi yang lebih besar dibandingkan dengan data model. Sedangkan uji coba ke tiga memiliki nilai yang lebih buruk dikarenakan terdapat kesalahan pada tangan kanan dan kiri pengguna. Dari hasil uji coba ini dapat dilihat kemampuan program dalam menilai posisi tubuh dimana aplikasi mampu memberikan penilaian posisi tubuh yang salah dan memberikan penilaian yang memiliki korelasi antara data model dengan data pengguna aplikasi. Program mampu menilai posisi tubuh secara komprehensif.