BAB IV

IMPLEMENTASI DAN PENGUJIAN

IV.1 Implementasi

IV.1.1 Implementasi Perangkat Keras

Perangkat keras yang digunakan untuk menjalankan sistem pengontrol pointer mouse dengan hand tracking haruslah sebuah komputer yang memiliki spesifikasi cukup bagus dalam melakukan komputasi, hal ini dikarenakan akan adanya banyak proses perhitungan/komputasi dalam sistem tersebut, namun sebagai gambaran berikut sistem minimal yang dapat dipakai :

1. Processor : Intel Pentium 4
2. RAM : 384 MB
3. Space Hardisk : 15 GB
4. VGA minimal 128 MB
5. *Web-cam* (minimal 3MP)
6. Koneksi internet (dalam proses instalasi)

IV.1.2 Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan untuk membangun dan menjalankan sistem pengontrol pointer mouse dengan hand tracking adalah sebagai berikut :

1. Sistem Operasi Ubuntu Natty Narwhal (11.04) dan Ubuntu Maverick Meerkat(10.10)
2. GCC (GNU Compiler Cpllection) versi 6-dev

3. Opencv 2.2
4. QT Framework (minimal versi .7)
5. IDE Eclipse Helios
6. IDE QT Creator

IV.1.3 Implementasi Class Pada Sistem

IV.1.3.1 Implementasi Kelas ConContour

Berikut adalah implementasi class ConContour pada sistem yang dibangun:

```
class ConContour {

private:
	IColorSkinDetector *skinDetection;
	IplImage *img;
	IplImage *currentFrame;
	IplImage *tempImage;
	IplImage *skinTemp;

	IplImage *skin;
	IplImage *pyrdA;
	IplImage *pyrdB;
	IplImage *eig_image;
	IplImage *tmp_image;
	CvPoint2D32f *cornersA;

	CvSize img_sz;

	CvSize pyr_sz;
	IplImage *tempPyrA;
	IplImage *tempPyrB;
	CvPoint2D32f *cornersB;

	int win_size;

	char features_found[MAX_CORNERS];
	float feature_errors[MAX_CORNERS];
	//end object for lucas kanade
```

```
	int frameWidth;
	int frameHeight;

	CvScalar hsvMin;
	CvScalar hsvMax;
	CvScalar yccMin;
	CvScalar yccMax;

	CvSeq *hull;
	CvSeq *filtereHull;
	CvSeq *defect;

	CvPoint *arrHull;
	CvConvexityDefect *defectArray;
	CvRect *handRect;

	CvBox2D box;

	CvFont font;

	char number;

	CvMemStorage *gStorage;

	CvSeq *biggestContour;

	CvSeq *currentContour;
	CvSeq *contour;
	CvPoint startPoint;
	CvPoint depthPoint;
	CvPoint endPoint;

	CvPoint *setPoint;

	int fingerNum;

	  CvPoint p0;
	  CvPoint p1;

	  CvPoint p0S;
	  CvPoint p1S;
	  int timeFinger;
	  int tempFinger;


private:
	void drawAndComputeFingerNum();
	void checkGStorage(int i);

public:
	ConContour();
	~ConContour();
	void lukas_canade();
```

```
      void frameGrabber();

      void extractContourAndHull(IplImage *skin);

      IplImage *getCurrentFrame();

      void setCurrentFrame(IplImage *image);
      IplImage *getSkin();
      IplImage *getImg();
      int getFingerNum();
      CvPoint getP0();
      CvPoint getp1();
      //void clean();
};
```

IV.1.3.2 Implementasi Kelas ConHistogram

Berikut adalah implementasi kelas ConHistogram pada sistem yang dibangun:

```
class ConHistogram {
public:
      IplImage *frame;
      IplImage *singleFrame;
      IplImage *histimg;

      CvHistogram *hist;

      int hdims;
      float hranges_arr[2];
      float* hranges;
      int bin_w;
      float max_val;
      int i;
      double val;
      CvScalar color;

public:
      ConHistogram();
      ~ConHistogram();
      //accessor
      IplImage *getFrame();
      CvHistogram *getHist();
      IplImage *getHistimg();
      IplImage *getSingleFrame();
      void setFrame(IplImage *frame);
      void setHist(CvHistogram *hist);
```

```
      void setHistimg(IplImage *histimg);
      void setSingleFrame(IplImage *singleFrame);

public:
      //main methode
      void progressImage();
      void drawHistogram();
};
```

IV.1.3.3 Implementasi Kelas CounturWindow

Berikut adalah implementasi class ContourWindow pada sistem yang dibangun:

```
class CounturWindow : public QWidget
{
    Q_OBJECT
    private:
        QCounturWidget *widgetCountur;
        CvCapture *camera;
        IplImage *image;
        int set;

    public:

        CounturWindow(QWidget *parent=0);
        ~CounturWindow();
        void setImage(IplImage *image);
        void setSet(int s);

    protected:
        void timerEvent(QTimerEvent*);
};
```

IV.1.3.4 Implementasi Kelas DialogCountur

Pada implementasi class DialogContour yang diterlihat hanya satu kelas yang memiliki hubungan relasi dengan kelas ui_dialogcontour karena dalam Qt tidak dibenarkan mengakses langsung pada kelas inti user interface, sehingga digunakan kelas perantara sebagai berikut:

```
class DialogContour : public QDialog
{
    Q_OBJECT

public:
    explicit DialogContour(QWidget *parent = 0);
    ~DialogContour();
    void setCountur(IplImage *image, int i);

private slots:
    void on_buttonTutup_clicked();

private:
    Ui::DialogContour *ui;
};
```

IV.1.3.5 Implementasi Kelas DialogHistogram

Pada implementasi class DialogHistogram hamper sama dengan DialogContour yang diterlihat hanya satu kelas yang memiliki hubungan relasi dengan kelas ui_dialogcontour karena dalam Qt tidak dibenarkan mengakses langsung pada kelas inti user interface, sehingga digunakan kelas perantara sebagai berikut:

```
class DialogHistogram : public QDialog
{
    Q_OBJECT

public:
    explicit DialogHistogram(QWidget *parent = 0);
    ~DialogHistogram();
     void setQwidgetVal(IplImage *img,int val);

private slots:
     void on_buttonTutup_clicked();

private:
    Ui::DialogHistogram *ui;
};
```

IV.1.3.6 Implementasi Kelas HandTracking

Berikut adalah implementasi class HandTracking pada sistem yang dibangun:

```
class HandTracking {

private:

      Display *dpy;
      int searchx;
      int searchy;

      CvPoint p0;
      CvPoint p1;

      int stateX;
      int stateY;
      int konX;
      int konY;
      int countx;
      int county;

      int state;
      int redx;
      int redy;
      int starth;

public:
      HandTracking();
      ~HandTracking();

      void setDefault();
      void setState(int state);
      void setP0(CvPoint point);
      void setP1(CvPoint point);
      void on_mouse(int event, int x, int y, int flags, void*
param);

      void trackPoint(Display *dpy);
      void play();
      void getResult();
};
```

IV.1.3.7 Implementasi HistogramWindow

Berikut adalah implementasi class HistogramWindow pada sistem yang dibangun:

```
class HistogramWindow : public QWidget
{
    Q_OBJECT
    private:
        QHistogramWidget *widgetHistogram;

        ConHistogram *histogram;
        IplImage *img;
        int set;

    public:

        HistogramWindow(QWidget *parent=0);
        void setSet(int s);
        void setImage(IplImage *image);

    protected:
        void timerEvent(QTimerEvent*);
};
```

IV.1.3.8 Implementasi Kelas Image

Berikut adalah implementasi class Image pada sistem yang dibangun:

```
template<class T> class Image {
private:
      IplImage *imgp;
public:
      Image(IplImage *img = 0) {
            imgp = img;
      }
      ~Image() {
            imgp = 0;
      }
      void operator=(IplImage *img) {
            imgp = img;
      }
      inline T* operator[](const int rowIndex) {
            return ((T*) (imgp->imageData + rowIndex * imgp-
>widthStep));
      }
};
```

IV.1.3.9 Implementasi Kelas Listener

```
class listener
{
public:
            virtual void onStart()=0;
            virtual void onStop()=0;
            virtual void onHistogram()=0;
            virtual void onContour()=0;
            virtual void loadCountur(IplImage *image)=0;
            virtual void sinkronisasi(IplImage *image)=0;
            virtual void onTracking(CvPoint point, int state)=0;
            virtual void onPetunjuk()=0;
};
```

IV.1.3.10 Implementasi Kelas MyCameraWindow

Berikut adalah implementasi class MyCameraWindow pada sistem yang dibangun:

```
class MyCameraWindow : public QWidget
{
    Q_OBJECT
    private:
        QOpenCVWidget *cvwidget;
        CvCapture *camera;

        IplImage *image;
        IplImage *black;
        IplImage *gray;
        ConContour *cont;
       // HandTracking *tracking;
        SistemModel *model;
        int state;
        int histogram;
        int countur;

    public:
        MyCameraWindow( QWidget *parent=0);

        void setModel(SistemModel *model);
        void setState(int state);
        void setHistogram(int histogram);
        void setContour(int conrtour);
        int getState();
        int getHistogram();
        int getContour();

    protected:
        void timerEvent(QTimerEvent*);
};
```

IV.1.3.11 Implementasi SistemView

Berikut adalah implementasi class SistemView pada sistem yang dibangun:

```
class SIstemView : public QWidget
{
    Q_OBJECT

public:
    explicit SIstemView(QWidget *parent = 0);
    ~SIstemView();

private:
    InheritSistemView *ui;
};
```

IV.1.3.12 Implemenetasi KelasQOpenCvWidget

Berikut adalah implementasi class KealsQOpenCvWidget pada sistem yang dibangun:

```
class QOpenCVWidget : public QWidget {
    private:
        QLabel *imagelabel;
        QVBoxLayout *layout;

        QImage image;

    public:
        QOpenCVWidget(QWidget *parent = 0);
        ~QOpenCVWidget(void);
        void putImage(IplImage *);
};
```

IV.1.3.13 Implementasi Kelas YCrCbDetector

Berikut adalah implementasi class YCrCbDetector pada sistem yang dibangun:

```
class YCrCbSkindetector: public IColorSkinDetector {
public:
      IplImage *currentYCrCbFrame;
      IplImage *skin;
      IplConvKernel *rect_12;
      IplConvKernel *rect_6;

public:

      YCrCbSkindetector();
      IplImage *detectSkin(IplImage *img, CvScalar min, CvScalar
max);
};
```

IV.1.3.14 Implementasi Kelas InheritanceView

Berikut adalah implementasi class InheritanceView pada sistem yang dibangun:

```
class InheritSistemView : public QWidget, public
Ui::SIstemView,public listener
{
    Q_OBJECT
public :
    SistemModel *model;
    SistemControll *controll;
    HandTracking *tracking;

    DialogHistogram *histogram;
    DialogContour *countur;
    IplImage *hist;
    IplImage *cou;
    DialogHelp *help;

public:
    explicit
    InheritSistemView(QWidget *parent = 0);
    ~InheritSistemView();

public :
        void onStart();
        void onStop();
        void onHistogram();
        void onContour();
        void onTracking(CvPoint point, int state);
        void sinkronisasi(IplImage *image);
        void loadCountur(IplImage *image);
        void onPetunjuk();

public slots:
                void on_buttonStart_clicked();
                void on_buttonStop_clicked();
                void on_buttonHistogram_clicked();
                void on_buttonContour_clicked();
                void on_buttonExit_clicked();
                void on_buttonPetunjuk_clicked();
};
```

IV.1.3.15 Implementasi Kelas SistemModel

Berikut adalah implementasi class SistemModel pada sistem yang dibangun:

```
class SistemModel
{
private :
    int ptX;
    int ptY;
    CvPoint point;
    listener *listen;
    IplImage *image;
    IplImage *countur;
    int state;
public:
    SistemModel();

    void setPtX(int ptX);
    void setPtY(int ptY);
    int getPtX();
    int getPtY();
    void setImage(IplImage *image);
    void setCountur(IplImage *countur);
    void setPoint(CvPoint points);
    CvPoint getPoint();
    void setState(int state);
    int getState();
    void setListener(listener *list);

   void doHandTracking();
   void doStopHandTracking();
   void getHistogram();
   void getContour();
   void setSingkronisasiMouse();
   void singkronisasiHis();
   void loadCountur();
   void tampilPetunjuk();

protected :
     void fireOnStart();
     void fireOnStop();
     void fireOnHistogram();
     void fireOnContour();
     void fireOnSingkronisasi();
     void fireOnTracking();
     void fireOnLoadCountur();
     void fireOnPetunjuk();
};
```

IV.1.3.16 Implementasi Kelas SistemControll

Berikut adalah implementasi class SistemControll pada sistem yang dibangun:

```
class SistemControll
{
   private :
    SistemModel *model;

public:
    SistemControll();
    void setModel(SistemModel *model);
    void startTracking();
    void stopTracking();
    void startHistogram();
    void startContour();
    void showPetunjuk();
};
```

IV.1.4 Implementasi Konfigurasi *Library* OpenCV

OpenCV merupakan *library* untuk membantu dalam pembangunan sistem dengan teknologi komputer vision dengan menggunakan bahasa C, C++, dan python. Dalam mengimplementasikan sistem yang telah dibangun dibutuhkan *library* opencv yang dikonfigurasi kedalam sistem operasi agar sistem dapat menemukan *library* yang dibutuhkan.

Dalam proses instalasi opencv kedalam sistem dibutuhkan *dependency package* untuk mendukung *library* dari opencv tersebut, seperti *library* ffmpeg untuk format video terutama QT Framework yang merupakan framework untuk membangun GUI dari sistem. Berikut adalah daftar *depedency package* yang dibutuhkan :

1. cmake
2. libtbb2
3. libtbb-dev
4. qt
5. glib
6. ffmpeg
7. libgstreamer
8. libv4l
9. libxine
10. unicap
11. libdc1394
12. libavformat-dev libswscale-dev
13. libjpeg
14. libtiff
15. libjasper
16. libpng
17. zlib
18. openexr
19. gtk+-2.0
20. gthread-2.0
21. gstreamer-base-0.10

22. gstreamer-app-0.10
23. gstreamer-video-0.10
24. libavcodec
25. libavformat
26. libavutil
27. libswscale

Library tersebut harus terpasang sebelum opencv di install dan dikonfigurasi. Dalam sistem operasi ubuntu *library* ini dapat didownload dengan menggunakan fasilitas *apt* ataupun *synaptic manager*.

Berikut konfigurasi yang harus dilakukan ketika menginstall opencv :

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON
WITH_QT=ON ..

export LD_LIBRARY_PATH =~/projects/opencv/release/lib:
$LD_LIBRARY_PATH

sudo ldconfig
```

Karena banyaknya depedensi yang di butuhkan sehingga dibuat sebuah *shell script* yang berisi perinta untuk menginstall depdensi, opencv dan mengkonfigurasi opencv dengan mudah hanya dengan satu kali perintah. Shell script tersebut diberi nama *setupopencv.sh*, berikut adalah skrip shell yang dibuat :

```
#!/bin/bash

cd ~
mkdir ~/projects

cp OpenCV-2.2.0.tar.bz2 ~/projects
cd ~/projects
tar -xvf OpenCV-2.2.0.tar.bz2
mv OpenCV-2.2.0 opencv
cd ~/projects/opencv

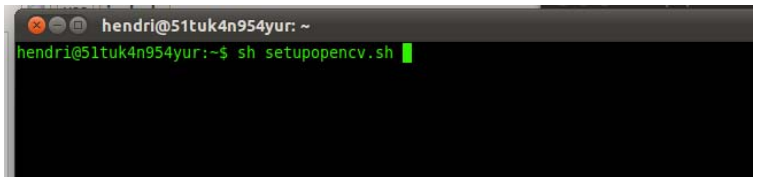
sudo apt-get install cmake
sudo apt-get install libtbb2
sudo apt-get install libtbb-dev
sudo apt-get install qt
sudo apt-get install glib
sudo apt-get install ffmpeg
sudo apt-get install libgstreamer
sudo apt-get install libv4l
sudo apt-get install libxine
sudo apt-get install unicap
sudo apt-get install libdc1394
sudo apt-get install libavformat-dev libswscale-dev
sudo apt-get install libjpeg
sudo apt-get install libtiff
sudo apt-get install libjasper
sudo apt-get install libpng
sudo apt-get install zlib
sudo apt-get install openexr
sudo apt-get install gtk+-2.0 gthread-2.0 gstreamer-
base-0.10 gstreamer-app-0.10 gstreamer-video-0.10
libavcodec libavformat libavutil libswscale

cd ~/projects/opencv
mkdir release
cd release

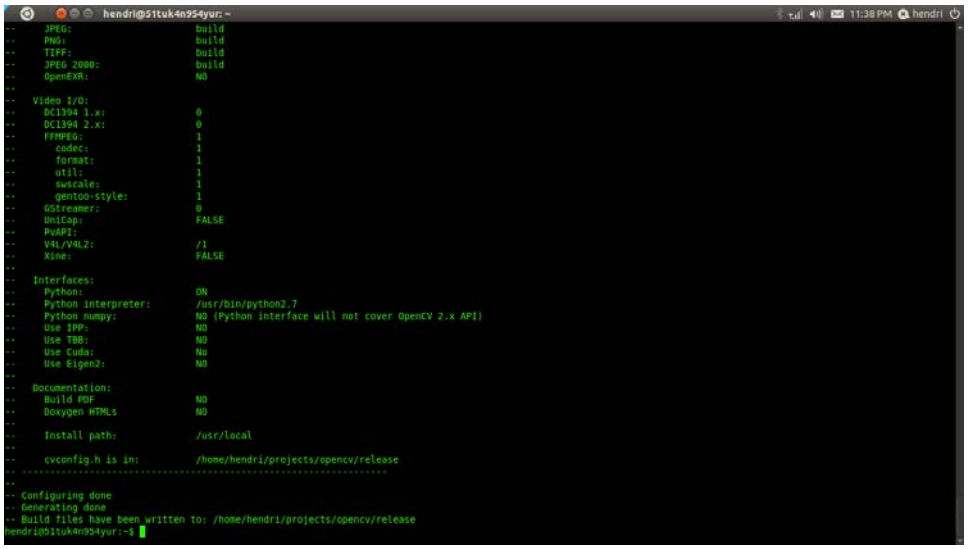
cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local -D
BUILD_PYTHON_SUPPORT=ON WITH_QT=ON ..

export
LD_LIBRARY_PATH=~/projects/opencv/release/lib:$LD_LI
BRARY_PATH
```

Sebelum menggunakan script tersebut, terlebih dahul copy file *OpenCV-2.2.0.tar.bz2* ke dalam direktori *home* pengguna, misalkan /home/hendri/. Setelah itu *copy* juga script setupopencv.sh kedalah direktori *home* yang sama. Setelah itu pengguna dapat menggunakan script tersebut dan mengeksekusi skrip setupopencv.sh. Skrip tersebut dapat dipanggil dengan menggunakan terminal atau menggunakan klik kanan lalu *run in terminal*. Maka skrip tersebut akan melakukan pencarian *package* pada server ubuntu dan menginstalkannya pada sistem operasi pengguna melalui koneksi internet.



Gambar IV.1 Perinta untuk mengeksekusi skrip setupopencv.sh



Gambar IV.2 Setelah berhasil terinstall dan dikonfigurasi

Setelah itu pengguna dapat menggunakan sistem pengontrol mouse pointer dengan *hand tracking* dengan memanggil file eksekusi dari sistem dengan nama file *main.*

IV.1.5 Implementasi Antarmuka Sistem

Saat pengguna menjalankan sistem, maka antarmuka yang ditampilkan oleh sistem adalah sebagai seperti pada gambar IV.3.



Gambar IV.3 Tampilan awal sistem ketika berjalan

Ketika tombol start ditekan atau menekan *shortcut* ctrl+A maka sistem akan langsung melakukan komputasi dan melakukan singkronisasi kondisi tangan dengan mouse pointer.

Untuk melakukan interaksi *event click* kanan atau kiri dapat menggunakan perubaha jumlah jari dari tangan pengguna.

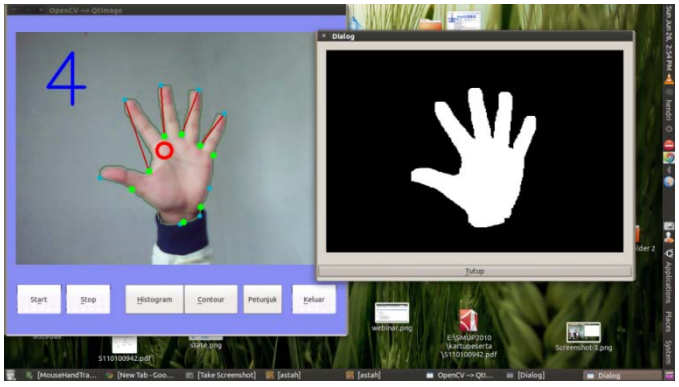
1. Untuk click kiri : 0 <= jumlah jari <= 1
2. Untuk click kanan : jumlah jari = 2.
3. Untuk keadaan normal : 3<=jumlah jari <=5

Ketika tombol Histogram diclick atau dengan *shortcut* ctrl+H maka fungsi histogram berjalan maka akan tampil kondisi histogram pada kondisi citra yang sedang dilihat oleh komputer, seperti yang ada pada gambar IV.5.



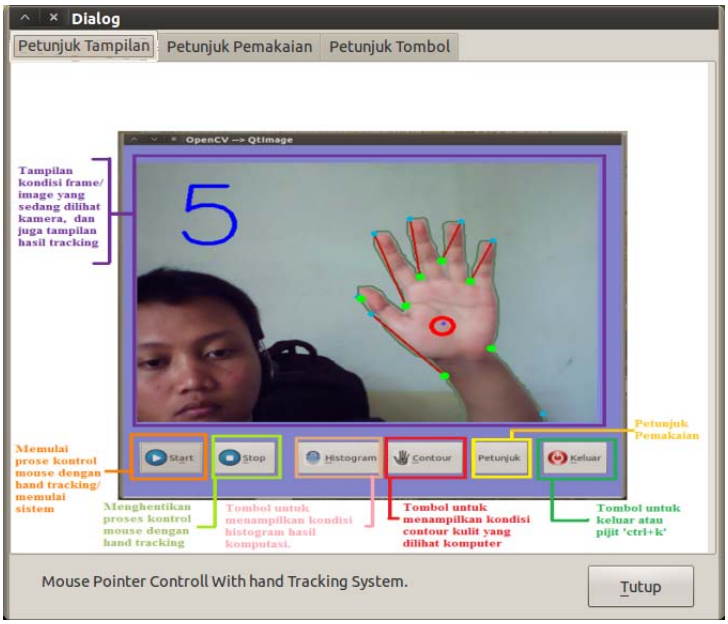
Gambar IV.4 Menampilkan form histogram

Ketika tombol Contour diclick atau dnegan *shortcut* ctrl+C maka fungsi untuk menampilkan kondisi contour dari citra yang sedang dilihat oleh komputer akan ditampilkan, seperti yang ada pada gambar IV.6.



Gambar IV.5 Form Contour ditampilkan

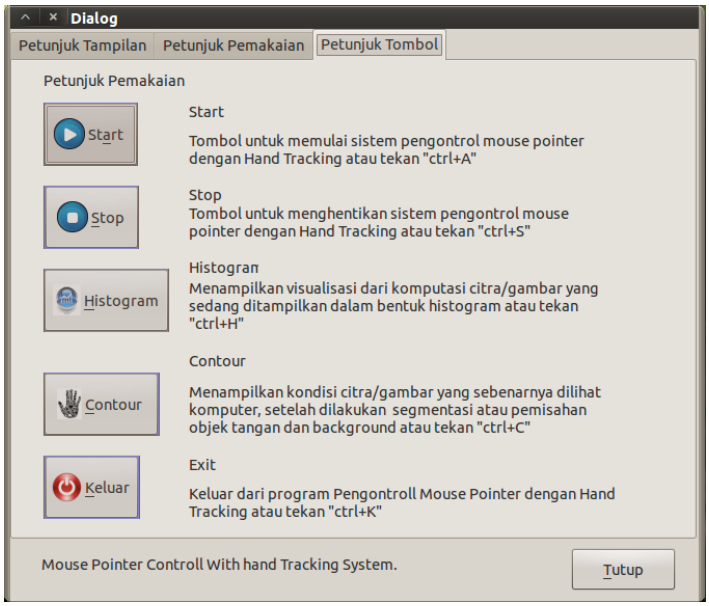
Disdiakan juga tombol/*button* petunjuk untuk menjelaskan cara pemakaian sistem dengan berbentuk form yang berisi gambar penjelasan mengenai fasilitas yang disediakan, seperti pada gambar IV.11, IV.12, dan IV.13.



Gambar IV.6 Tampilan Dialog Petunjuk untuk Tab 1 (Petunjuk Tampilan)



Gambar IV.7 Tampilan Dialog Petunjuk untuk Tab 2 (Petunjuk Pemakain Gestur)



Gambar IV.8Tampilan Dialog Petunjuk untuk Tab 3 (Petunjuk Tombol)

Jika tombol Stop diklik atau menggunakan shortcut ctrl+S maka proses utama atau proses komputasi dan singkronisasi akan dihentikan. Dan tombol Keluar jika diclick atau menggunakan shortcut ctrl+K, maka sistem akan berhenti dan atau keluar dari sistem.

IV.2 Pengujian

IV.2.1 Pengujian Alpha

IV.2.1.1 Skenario Pengujian

Skenario pengujian menjelaskan skenario pengujian terhadap sistem pengontrol mouse pointer komputer dengan hand tracking.

A. Skenario Pengujian Sistem Pada Kelas Utama Untuk Menampilkan Citra

Tabel IV-1 Tabel butir uji sistem pada kelas utama untuk menampilkan citra

Kelas Uji	Butir Uji	Jenis Pengujian
Utama/QOpenCVWidget	Pendeteksi Kamera Pengambilan Citra hasil komputasi	White Box
ConCountur / Kelas Pendeteksi Tangan Contour Tanga	Proses komputasi untuk segmentasi warna kulit	White Box
	Proses komputasi penggambaran kontur tangan, ConvexHull dan Convexity Defects	
	Perhitungan jumlah jari pada objek dari citra yang ditangkap	
	Penempatan Algoritma pyramidal lucas-kanade	

IV.2.1.2 Butir Pengujian

Pengujian dilakukan dengan menguji setiap proses komputasi utama yang didalam *sistem* yang terdiri dari dua kelas yaitu kelas QOpenCVWidget dan kelas ConContour. Jika pada kelas QOpenCVWidget terjadi kesalahan utama yaitu pada saat mendeteksi kamera dan ternyata kamera tidak ada maka seluruh proses tidak dapat berjalan, namun apabila kamera dapat digunakan dan terjadi kesalahan lain pada saat komputasi misalkan tidak ada *input* citra yang memiliki objek tangan pada saat sistem berjalan maka tidak akan terjadi proses komputasi maka sehingga akan menghemat penggunaan memori, namun jika terjadi kesalahan yang di akibatkan habisnya memori yang ada pada sistem operasi atau terjadi kegagalan alokasi memori maka sistem akan mengeluarkan *warning* dan akan segera keluar dari sistem karena tidak dapat mengalokasikan memori.

Pengujian yang dilakukan menggunakan pengujian *white box* yaitu dengan menguji setiap prosedur komputasi ataupun proses validasi yang harus dilewati untuk memastikan proses tersebut dapat berjalan secara efisien dan semua proses memiliki fungsi yang tepat guna dengan menggunakan sekumpulan dasar dari alur eksekusi.

Penentuan nilai *cyclomatic complexity* untuk diagram alir dengan menggunakan rumus :

$$V(G) = E - N + 2 \quad \text{.....(35)}$$

V(G) = *cyclomatic complexity*

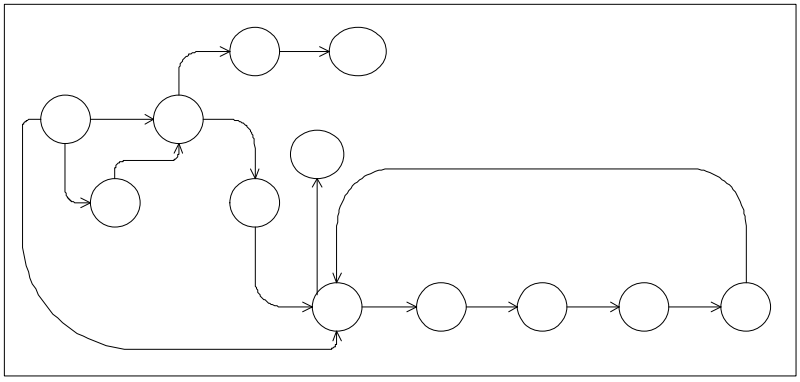
E = Jumlah Edge pada grafik alir

N = Jumlah Edge pada grafik alir

IV.2.1.2.1 Butir Pengujian Sistem Kelas Utama Pendeteksi Kamera Pengambilan Citra hasil komputasi

Penangkapan citra yang merupakan input yang diterima oleh sistem berupa citra tangan manusia atau anggota tubuh manusia dengan menggunakan perangkat keras kamera web terlebih dahulu harus dilakukan pendeteksian kamera itu sendiri. Pendeteksian yang dilakukan dengan melakukan pendeteksian kamera 0 hingga 1 (hanya 2 kamera) jika masing-masing tidak terdeteksi maka sistem tidak dapat berjalan. Pengujian untuk prosedur pengambilan citra dan juga pendeteksian kamera dapat dilihat pada gambar IV.9. Berikut adalah psudeocode prosedur timerSeq.

```
Procedure timerSeq
1 :  capture := cvGetFromCam(0)
2 :  if capture = Null then
3 :  capture := cvGetFromCam(1)
4 :  endif
5 :  if image = Null then
6 :  showMessage("Maaf kamera tidak ditemukan…");
7 :  end
8 :  endif
9 :  do while  start = true
10 : image := countur.getGraber(capture)
11 : tracking.setPos(counter.getPos)
12 : tracking.play()
13 : enddo
14 : end
```



Gambar IV.9 Flow Graph timerSeq prosedur menampilkan citra hasil komputasi dan pendeteksian kamera

Jumlah dari *cyclomatix complexity* atau V(G) dari diagram alir pada gambar IV.9 adalah sebagai berikut :

$$V(G) = 14 - 12 + 2 = 4$$

Dari gambar IV.9 didapat *independent path* sebagai berikut :

Path = 1-2-3-4-5-8-9-10-11-12-13-9-13

Path = 1-2-4-5-8-9-10-11-12-13-9-13

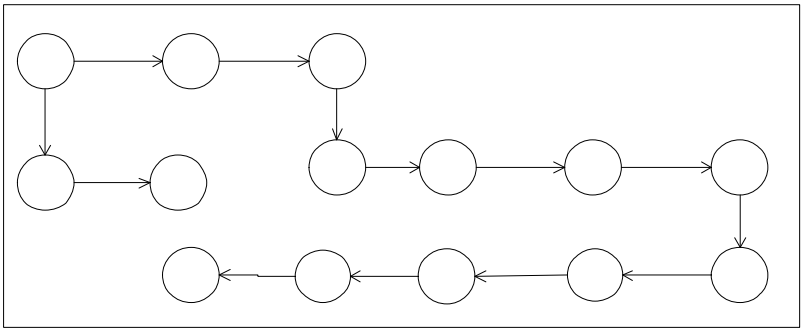
Path = 1-2-3-4-5-6-7

Path = 1-2-4-5-6-7

IV.2.1.2.2 Butir Pengujian Sistem pada Proses Komputasi untuk Segmentasi Warna Kulit

Setelah didapat input berupa citra dari setiap frame yang didapat maka citra tersebut perlu dilakukan proses segmentasi untuk memisahkan antara objek kulit (*foreground*) dan *background*. Walupun didalam citra tidak terdapat objek tangan atau bagian tubuh manusia, hasil result yang berupa objek-objek citra dan kontur yang bernilai Null dari proses ini tidak akan mempengaruhi sistem hingga mengeluarkan error, sistem akan melewati setiap komputasi yang tidak diperlukan jika dalam input yang didapat tidak terdapat objek bagian tubuh manusia. Pengujian untuk prosedur proses komputasi untuk segmentasi warna kulit dapat dilihat pada gambar IV.10. Berikut adalah psudeocode prosedur getGraber yang merupakan proses komputasi untuk segmentasi warna kulit.

```
Procedure getGraber()
1 :  image := cvGetQuery(capture)
2 :  if checkAllocation() = false then
3 :  showMessage(“Error : Memmory tidak mencukupi..”)
4 :  end
5 :  endif
6 :  cvConvertColor(image,imageCopy, CV_RGB_to_YCrCb)
7 :  cvInRangeS(currentYCrCbFrame, min, max, skin)
8 :  cvErode(skin, skin, rect_12, 1)
9 :  cvDilate(skin, skin, rect_6, 2)
10 : extractCounturAndHull()
11 : drawCounturAndCountFinger()
12 : cvCvtColor(imageCenter, pyrA, CV_RGB_to_GRAY)
13 : lucas_kanade()
14 : cvCopy(pyrA,pyrB)
15 : cvClear(imageCenter)
16 : return image
17 : end
```



Gambar IV.10 Flow Graph procedure getGraber yang mendeteksi warna kulit dan melakukan segmentasi lalu memanggil proses utama

Jumlah dari *cyclomatix complexity* atau V(G) dari diagram alir pada gambar IV.10 adalah sebagai berikut :

$$V(G) = 13 - 14 + 2 = 1$$

Dari gambar IV.10 didapat *independent path* sebagai berikut :

Path = 1-2-6-7-8-910-11-12-13-14-15-16

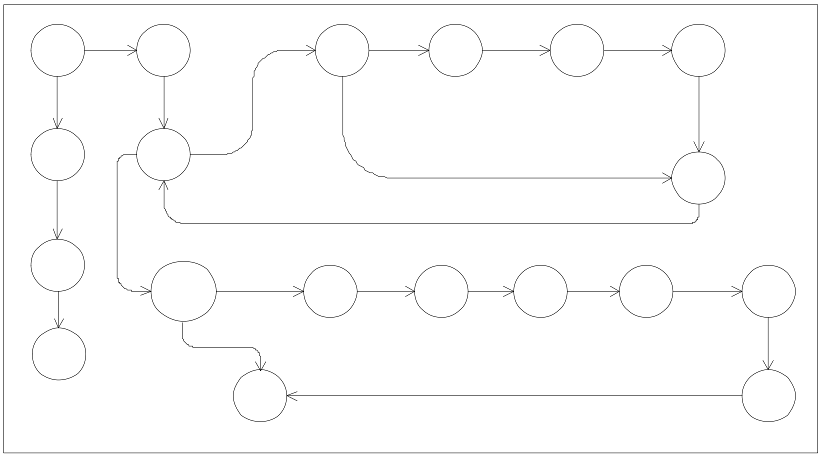
IV.2.1.2.3 Butir Pengujian Sistem pada Proses Komputasi Penggambaran Kontur Tangan, ConvexHull dan Convexity Defects

Proses komputasi utama adalah penggambaran kontur tangan, convexhull dan convexity defect. Penggambaran atau pencarian kontur tangan dibutuhkan citra hasil segmentasi dengan kedalaman warna 2 bit. Proses awal yang berjalan adalah menemukan kontur secara bebas lalu dilakukan pencarian kontur terbesar dalam arti kontur yang terlihat jelas. Setelah itu dilakukan pencarian hull dan defect dengan menggunakan metode convexhull dan convexity defect. Ketika objek tangan yang dicari dalam suatu citra tidak ditemukan dalam arti tidak ditemukan kontur terbesar yang terlihat jelas tidak ada maka proses convexity

defect akan dilewat atau tidak akan diproses untuk menghindari error yang mungkin terjadi karena tidak terdapat sedikit punkontur yang terlihat secara jelas.

Pengujian untuk prosedur proses komputasi penggambaran kontur tangan, convexHull dan convexity defects dapat dilihat pada gambar IV.10. Berikut adalah psudeocode prosedur extractCounturAndHull yang merupakan proses komputasi untuk menemukan kontur dan menemukan nilai hull dan defect.

```
Procedure extractCounturAndHull()
1 :  if checkAllocation() = false then
2 :  showMessage(“Error : Memmory tidak mencukupi..”)
3 :  end
4 :  endif
5 :  contour = cvFindCountur(skin,contour,storage)
6 :  do while contour <> Null
7 :  res1 = contour.toral
8 :  if res1 > res2 then
9 :  res2 = res1
10 : biggestContour = contour
11 : endif
12 : contour := contour.next
13 : enddo
14 : if biggestCountur <> Null then
15 : cvDrawCountur(biggestContour,storage)
16 : hull = convexHull(biggestContour)
17 : imageBox = cvMinRect(biggestContour,storage)
18 : imageCenter = createCircle(imageBox)
19 : defects = cvConvexityDefects(biggestContour, hull)
20 : arrDefect = cvCreateArray(defects, storage)
21 : endif
22 : end
```



Gambar IV.11 Flow Graph procedure extractCounturAndHull yang melakukan penggambaran kontur dan convexhull juga convexity defect

Jumlah dari *cyclomatix complexity* atau V(G) dari diagram alir pada gambar IV.11 adalah sebagai berikut :

$$V(G) = 21 - 20 + 2 = 3 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (36)$$

Dari gambar IV.11 didapat *independent path* sebagai berikut :

Path = 1-2-3-4
Path = 1-5-6-7-8-9-10-11-12-6-13-14-15-16-17-18-19-20-21
Path = 1-5-6-7-8-9-10-11-12-6-13-21

IV.2.1.2.4 Butir Pengujian Sistem pada Proses Penghitungan Jumlah Jari pada Objek dari Citra yang Ditangkap

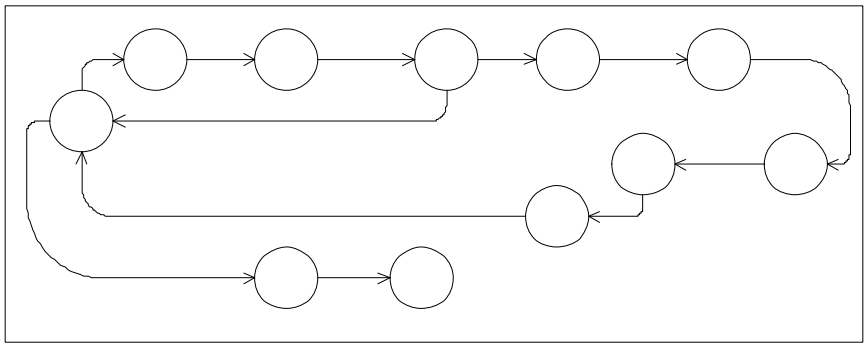
Hasil dari proses pencarian hull dan defect adalah kordinat dari setiap poin defect yang mengitari objek lengan dan berada pada setiap titik terluar dari objek yang mengitari tangan. Titik kordinat tersebut yang digunakan untuk menghitung banyaknya jari. Ketika tidak terdeteksi terdapat objek tangan atau bagian tubuh manusia pada citra maka array dari defect yang berisi titik kordinat defect bernilai

total 0, sehingga tidak memperngaruhi sistem secara keseluruhan dan tidak menimbulkan kegagalan sistem secara keseluruhan hanya pada jumlah jari akan mendefinisikan sebanyak 0.

Pengujian untuk prosedur proses penghitungan jumlah jari pada objek dari citra yang ditangkap dapat dilihat pada gambar IV.10. Berikut adalah psudeocode prosedur drawCounturAndCountFinger yang merupakan proses komputasi untuk menggambar defect dan menghitung jumlah jari yang terdeteksi.

```
Procedure drawCounturAndCountFinger()
1 :  For i=0 to defects.total do
2 :  startPoint = arrDefect.Start
3 :  endpoint = arrDefect.End
4 :  deptPoint = arrDdefect.Dept
5 :  if  ((startPoint.y  <  imageBox.y)  OR  (deptPoint.y<
     imageBox.y))  AND  (startPoint.y  <  depthPoint.y)  &&
     (sqrt(pow(startPoint.x – deptPoint.x,2) + pow(startPoint
     - deptPoint.y,2)) > (imageBox.height /6.0)) then

6 :  fingerNum = fingerNum + 1
7 :  cvLine(image,  startPoint,  depthPoint,  CV_RGB(200,10,0),
     2)
8 :  endif
9 :  cvCircle(image, startPoint, 2, CV_RGB(0,178,211), 5)
10 : cvCircle(image, depthPoint, 2, CV_RGB(0,178,211), 5)
11 : enddo
12 : end
```



Gambar IV.12 Flow Graph procedure drawCounturAndCountFinger merupakan proses kalkulasi jumlah jari yang terlihat pada objek dalam citra

Jumlah dari *cyclomatix complexity* atau V(G) dari diagram alir pada gambar IV.12 adalah sebagai berikut :

$$V(G) = 12 - 11 + 2 = 3$$

Dari gambar IV.12 didapat *independent path* sebagai berikut :

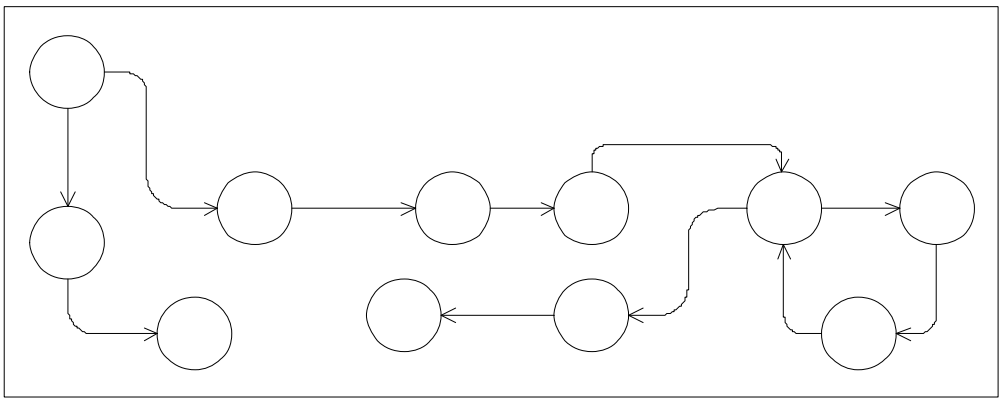
Path = 1-11-12
Path = 1-2-3-4-5-1-11-12
Path = 1-2-3-4-5-6-7-8-9-10-1-11-12

IV.2.1.2.5 Butir Pengujian Sistem pada Proses Penghitungan Jumlah Jari pada Objek dari Citra yang Ditangkap

Dari proses pencarian kontur terbesar dilakukan pencarian titik kordinat dari titik tengah kontur tangan atau anggota tubuh manusia yang ada dalam citra. Titik tersebut yang akan digunakan dalam metode utama yaitu pyramidal lucas-kanade. Algoritma pyramidal lucas kanade akan menghitung pergerakan dari titik yang merupakan titik tengah dari kontur tangan atau anggota tubuh manusia yang terlihat. Titik tersebut akan dikenakan komputasi untuk mencari sudut terbaik untuk dilakukan pencarian *pixel* yang berpindah. Dalam kondisi normal mudah dalam melakukan pendeteksian, namun ketika citra yang dilihat tidak terdapat kontur tangan ataupun anggota tubuh manusia, maka algoritma tersebut aktif namun tidak melakukan komputasi untuk mendeteksi *motion* atau pergerakan suatu *pixel* karena titik tersebut dihilangkan sehingga algoritma tidak dapat melihat *pixel* apapun yang bergerak.

Pengujian untuk prosedur proses metode pyramidal lucas-kanade pada objek dari citra yang ditangkap dapat dilihat pada gambar IV.13. Berikut adalah psudeocode prosedur lucas_kanade yang merupakan implementasi pyramidal lucas-kanade pada sistem terutama untuk mendeteksi pergerakan dari objek titik tengah *pixel* dari tangan manusia.

```
Procedure lucas_kanade()
1 :  Rest = InisialisasiVariable()
2 :  if Rest = false then
3 :  return 0
4 :  endif
5 :  cvGoodFeaturesToTrack(pyrdA, eig_image, tmp_image,
     cornersA)
6 :  cvFindCornerSubPix(pyrdB, cornersA, 20, 0.03)
7 :  cvCalcOpticalFlowPyrLK(pyrdA, pyrdB, tempPyrA, tempPyrB,
     cornersA,cornersB)
8 :  for I = 0  to I < cirber_count do
9 :  if (features_found[i] == 0) AND (feature_errors[i] >
     550) then
10 : p0S = cvPoint(cornersA[i].x, cornersA[i].y)
11 : p1S = cvPoint(cornersB[i].x, cornersB[i].y);
12 : enddo
13 : end
```



Gambar IV.13 Flow Graph procedure drawCounturAndCountFinger merupakan proses kalkulasi jumlah jari yang terlihat pada objek dalam citra

Jumlah dari *cyclomatix complexity* atau V(G) dari diagram alir pada gambar IV.13 adalah sebagai berikut :

$V(G) = 11 – 11 + 2 = 2$

Dari gambar IV.13 didapat *independent path* sebagai berikut :

Path = 1-2-3-4
Path = 1-2-5-6-7-8-9-10-11-8-12-13

IV.2.1.3 Kesimpulan Pengujian Alpha

Dari hasil pengujian yang didapatkan, dapat disimpulkan bahwa sistem sudah berjalan sesuai dengan yang diharapkan dan diperkirakan, proses penanganan kamera yang sudah berjalan dan proses komputasi guna mendeteksi objek tangan pada suatu citra ketika citra yang didapat bukan merupakan objek tangan ataupun bagian tubuh manusia maka proses komputasi tersebut tidak dilakukan sehingga dapat memperkecil penggunaan memori guna efisiensi penggunaan memori.

Secara fungsional sistem sudah dapat menghasilkan keluaran yang diharapkan penerapan algoritma lucas kanade berjalan sesuai dengan pergerakan yang dilakukan objek tangan atau bagian tubuh manusia sehingga mempermudah singkronisasi pointer mouse.