



Computer Vision

Hands on Lab



September 2013

For the latest information, please see bluejack.binus.ac.id



Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Binus University.

This document is for informational purposes only. BINUS UNIVERSITY MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Binus University.

Binus University may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Binus University, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2011 Binus University. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Table of Content

OVERVIEW	III
CHAPTER 01 INTRODUCTION TO FLTK AND OPENCV	1
CHAPTER 02 MORPHOLOGICAL FILTERING	21
CHAPTER 03 SEGMENTATION AND DETECTING USING HOUGH TRANSFORM	42
CHAPTER 04 PATTERN RECOGNITION AND CLASSIFICATION	68



OVERVIEW

Chapter 1

- Introduction to FLTK and OpenCV
Mengenal *library* FLTK dan openCV, serta dapat membuat program sederhana.

Chapter 2

- Morphological Filtering
Mempelajari jenis pemrosesan citra yang sederhana, meliputi *smoothing*, *grayscale*, dan *thresholding*.

Chapter 3

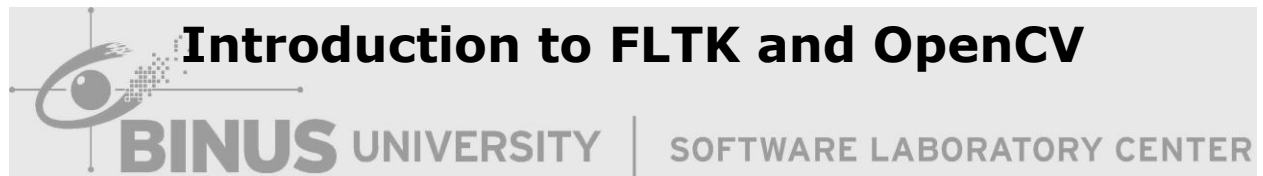
- Segmentation and Detecting Shape Using Hough Transform

Melakukan pemrosesan *image* untuk dapat mendeteksi sisi, menggunakan *canny*, *sobel* dan *laplacian edge detection*, serta dapat melakukan pendeteksian objek-objek sederhana, meliputi garis, lingkaran, dan persegi.

Chapter 4

- Pattern Recognition and Classification
Mengetahui suatu langkah awal pada *pattern recognition*, seperti deteksi wajah (*face detection*).

Chapter 01



1.1. FLTK

FLTK merupakan singkatan dari **Fast Light Toolkit**, adalah sebuah **GUI** (*Graphic User Interface*) *library* yang dapat bekerja di berbagai macam jenis *platform* (**OS**). Kelebihan **FLTK** dibanding *toolkit* lainnya (seperti **Qt** dan **WxWidget**) adalah kinerja desain yang lebih ringan dibandingkan dengan yang lain dan terbatas hanya untuk fungsionalitas **GUI** saja, sehingga ukurannya kecil dan terhubung secara statik dengan aplikasi yang dibuat.

1.2. OpenCV

OpenCV adalah sebuah *library* yang digunakan untuk memproses gambar yang dapat berjalan secara *multiplatform*.

Pengaplikasian **OpenCV** mencakup:

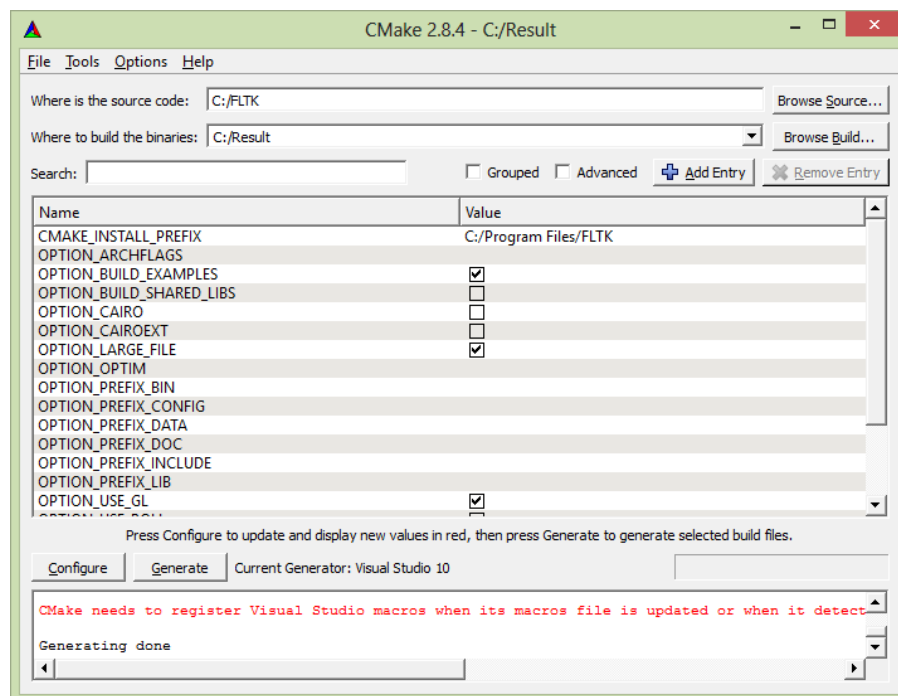
- *2D and 3D feature toolkits*
- *Ego-motion*
- *Face Recognition*
- *Gesture Recognition*
- *Human-Computer Interface (HCI)*
- *Mobile Robotics*
- *Motion Understanding*
- *Object Identification*
- *Segmentation and Recognition*
- *Stereopsis Stereo Vision: depth perception from 2 cameras*
- *Structure from Motion (SFM)*
- *Motion Tracking*



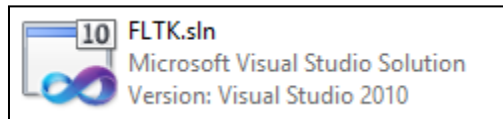
1.3. Instalasi FLTK dan OpenCV

Langkah-langkah untuk instalasi **FLTK** dan **OpenCV**, serta menghubungkan keduanya dengan **Microsoft Visual Studio 2010 C++** adalah sebagai berikut:

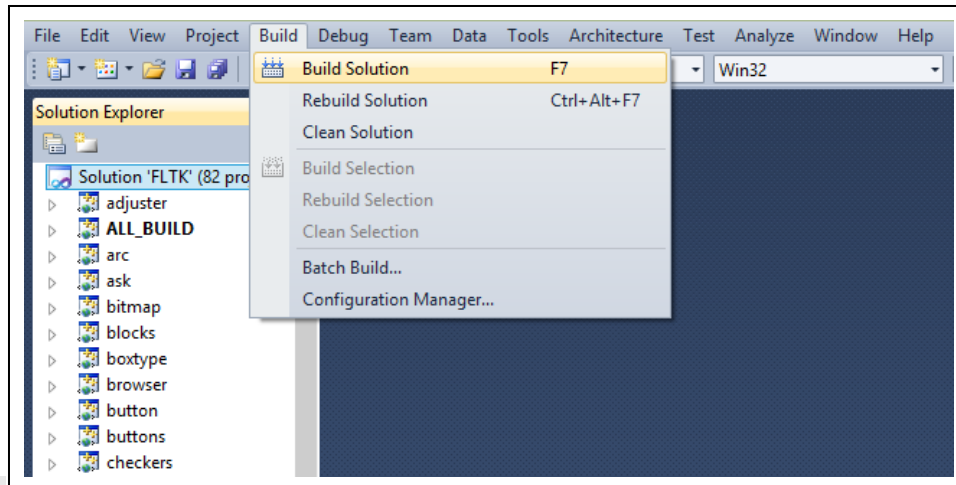
1. **FLTK** dapat di-*download* secara gratis melalui *website* resmi di <http://www.fltk.org> (versi yang paling stabil dan digunakan pada pembelajaran ini adalah **fltk1.3.2**). Setelah di-*download*, *extract*-lah **fltk-1.3.2-source.tar.gz** (lakukan *extract* 2 kali jika diperlukan), maka akan menghasilkan *folder* **fltk-1.3.2**. Untuk kemudahan, *rename*-lah *folder* **fltk-1.3.2** menjadi **FLTK**.
2. Pastikan **Microsoft Visual Studio 2010 C++** sudah ter-*install*. Kemudian, *install*-lah **CMAKE-2.8.4** untuk membuat **lib** terhadap **Visual Studio 2010**. Bukalah program **CMAKE**, kemudian **Browse Source** ke *folder* **FLTK**. **Browse Build** ke *folder* lain sebagai penampung hasil *build*, misalnya **C:/Result**. Klik **Configure**, kemudian pilih **Visual Studio 10** sebagai *generator*. Klik **Finish**. Setelah *load* konfigurasi sudah selesai, klik **Configure** sekali lagi untuk meng-*update* nilai yang berwarna merah. Kemudian, klik **Generate**, maka *code* **fltk** sudah di-*build*. Hasil *build* ada di **C:/Result**.



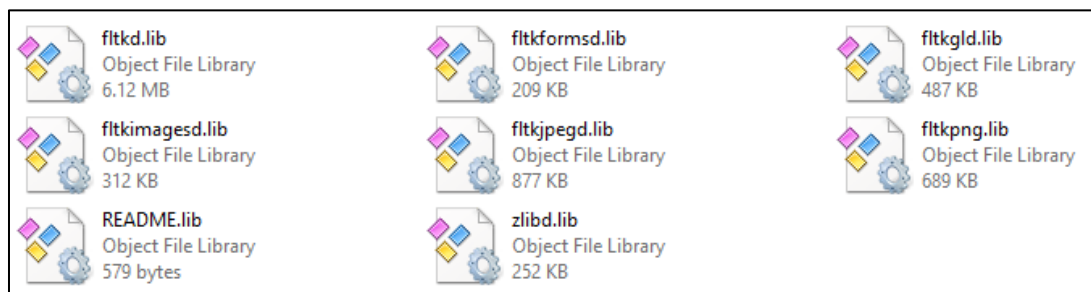
3. Di dalam *folder* **C:/Result**, akan ada *file* bernama **FLTK.sln**. Bukalah dengan **Visual Studio 2010** dengan cara *double-click*.



Kemudian, pada bagian **Menu**, pilih **Build > Build Solution**.



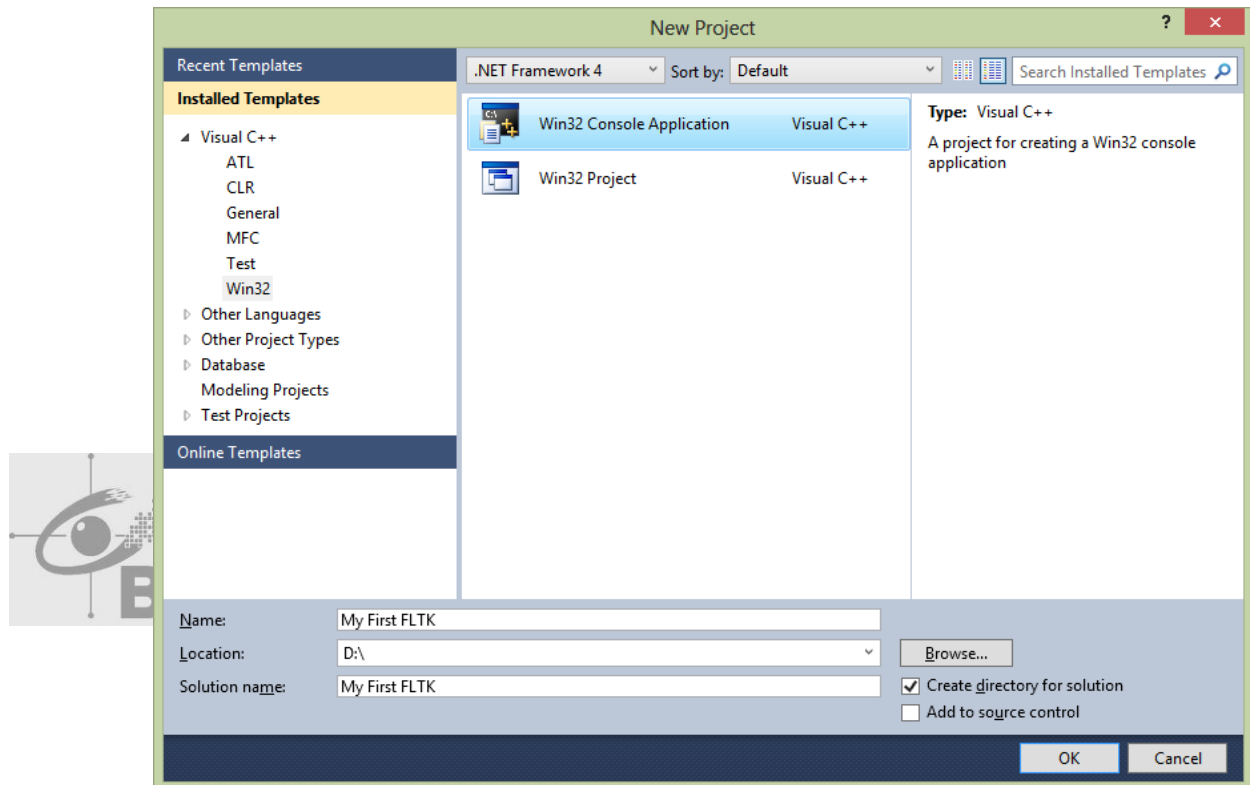
4. Setelah dilakukan **Build Solution**, maka pada *folder* **D:/Result/lib/Debug** akan terdapat *file* berekstensi **.lib**, seperti **fltkd.lib**, **fltkformsd.lib**, dan sebagainya. *Copy*-lah semua *file* berekstensi **.lib** ke dalam *folder* **D:/FLTK**. Maka **FLTK** sudah siap digunakan untuk **Visual Studio 2010**.



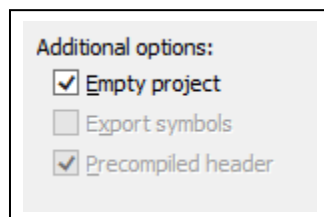
5. **OpenCV** yang digunakan adalah **OpenCV 2.3**. **OpenCV 2.3** dapat di-download di: <http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.3/>. (versi tersebut digunakan untuk **Windows OS**). Setelah di-download, *install*-lah dengan cara *double-click* pada **OpenCV-2.3.0-win-superpack.exe**, maka akan menghasilkan *folder* **OpenCV2.3**. Pada **OpenCV 2.3**, tidak diperlukan lagi *build solution* dengan

menggunakan **CMAKE**. Karena secara *default*, binary sudah disediakan untuk **Visual Studio 2010**.

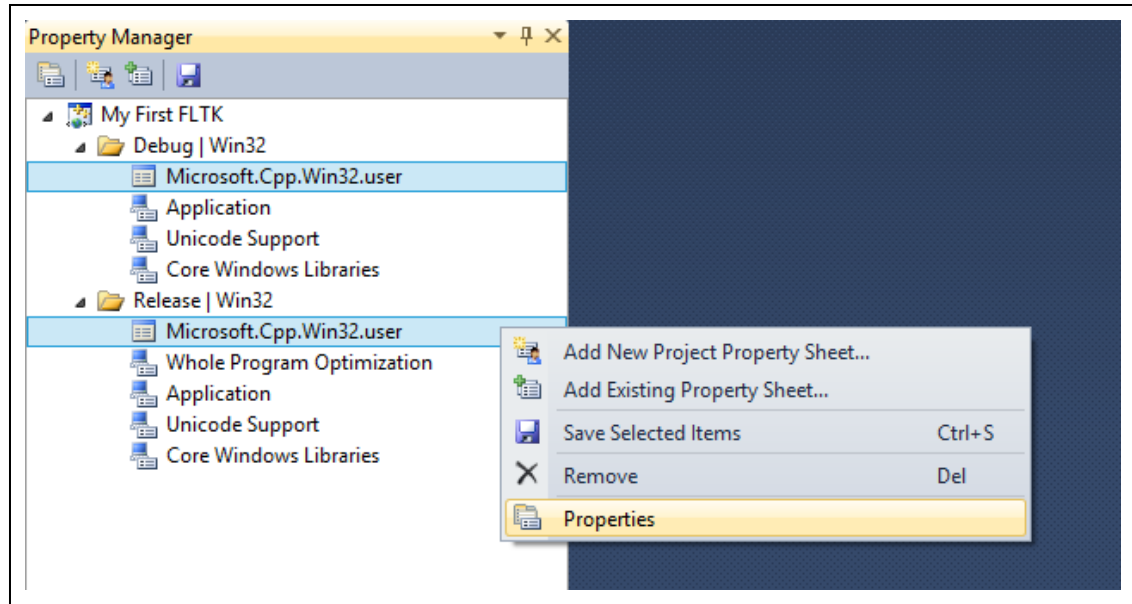
6. Buka **Microsoft Visual Studio 2010 C++**. Buatlah suatu *project* baru dengan cara **File > New > Project**. Pilih template **Win32** dengan aplikasi **Win32 Console Application**. Beri nama, misalnya “**My First FLTK**”.



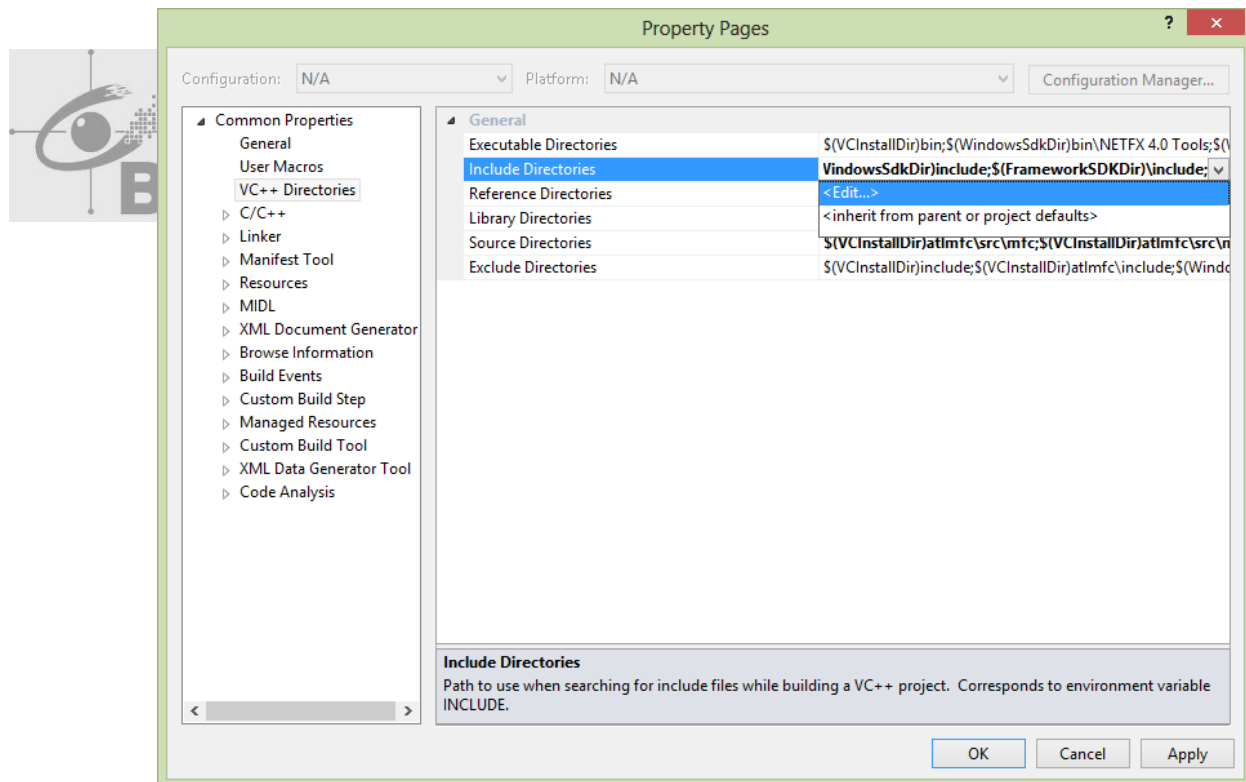
Klik **Next**, beri tanda *check* pada **Empty Project**, kemudian klik **Finish**.



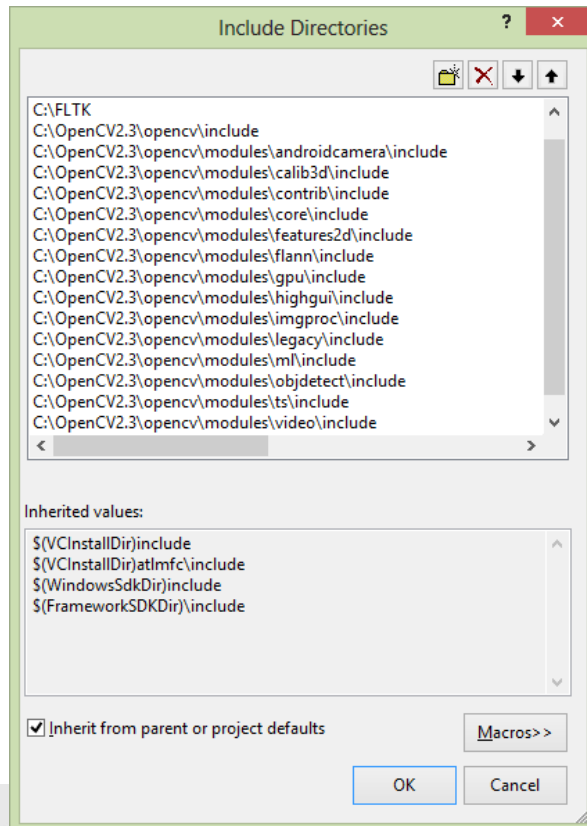
Kemudian, klik menu **View > Property Manager**. Pada *panel* **Property Manager**, klik **Microsoft.Cpp.Win32.user** yang ada di *folder* **Debug** dan **Release** pada *project* “**My First FLTK**” yang telah dibuat. Lalu klik kanan, dan pilih **Properties**.



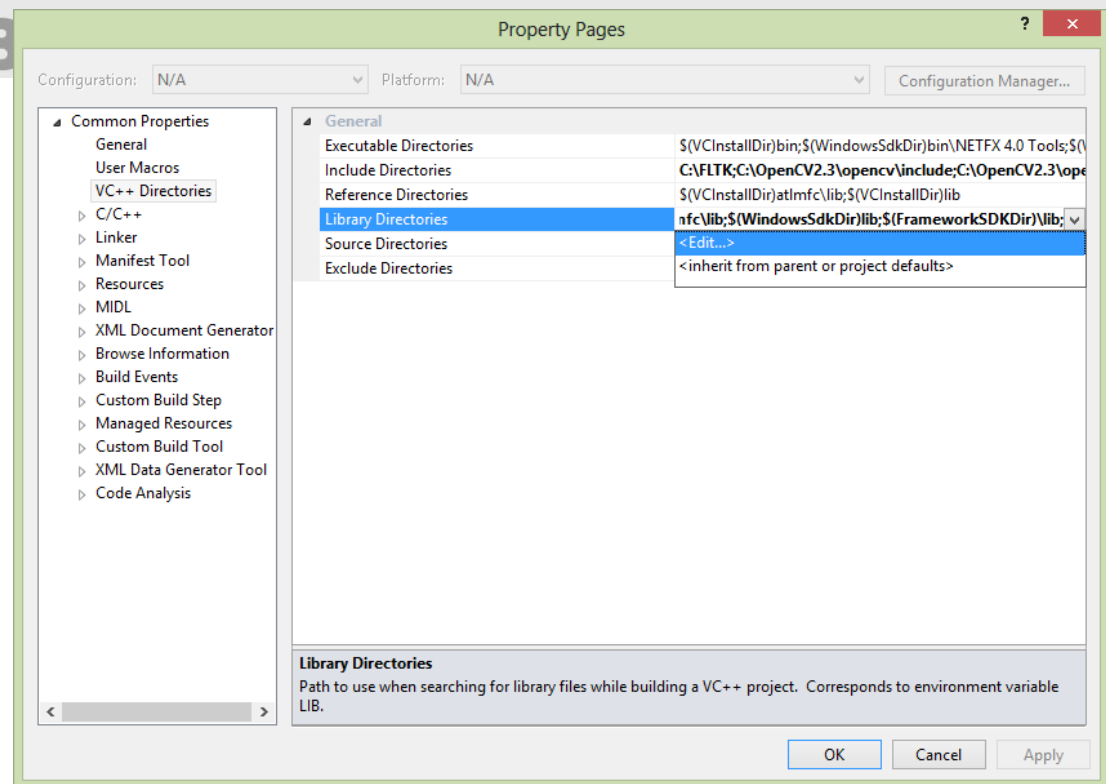
Pada bagian kiri, pilih **Configuration Properties > VC++ Directories**. Pada **Include Directories**, klik tanda **panah**, dan pilih **Edit**.



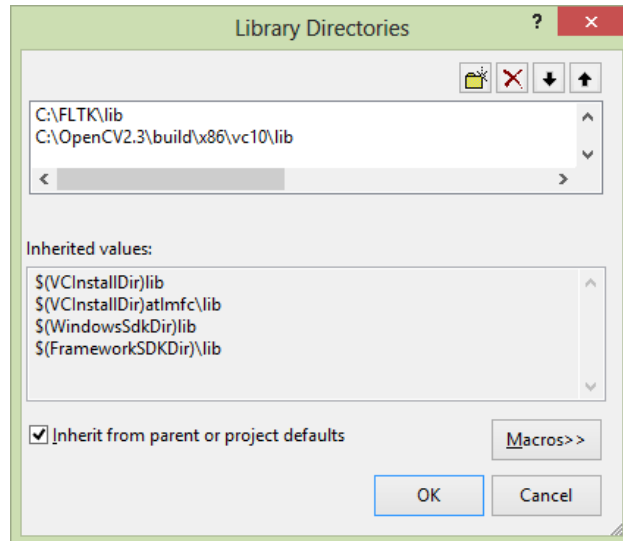
Kemudian, pada dialog **Include Directories**, masukkan *directory* untuk **include FLTK** dan **OpenCV** (dapat dilakukan dengan *browse*). Kemudian, klik **OK**.



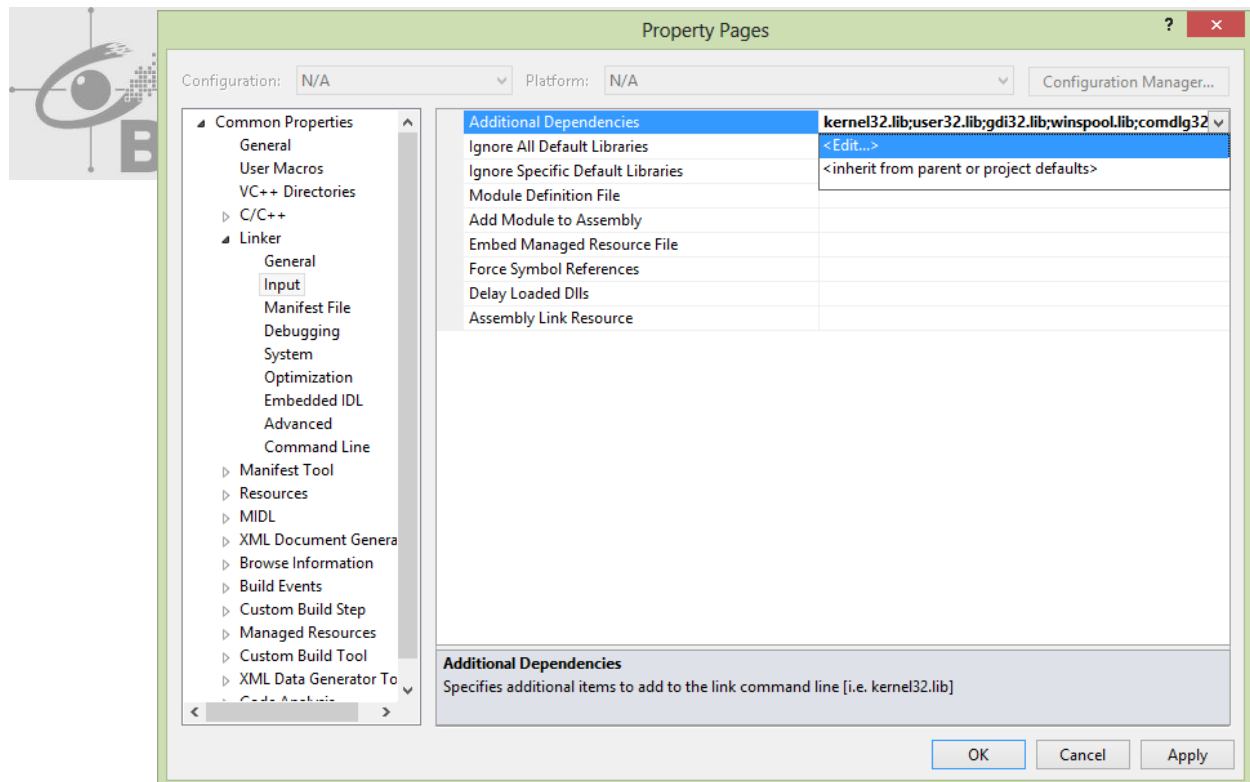
Masih pada **VC++ Directories**, pilih **Library Directories**, klik panah, pilih **Edit**.



Pada dialog **Library Directories**, masukkan dua direktori yang berisi **lib FLTK** dan **lib openCV**. Kemudian klik **OK**.

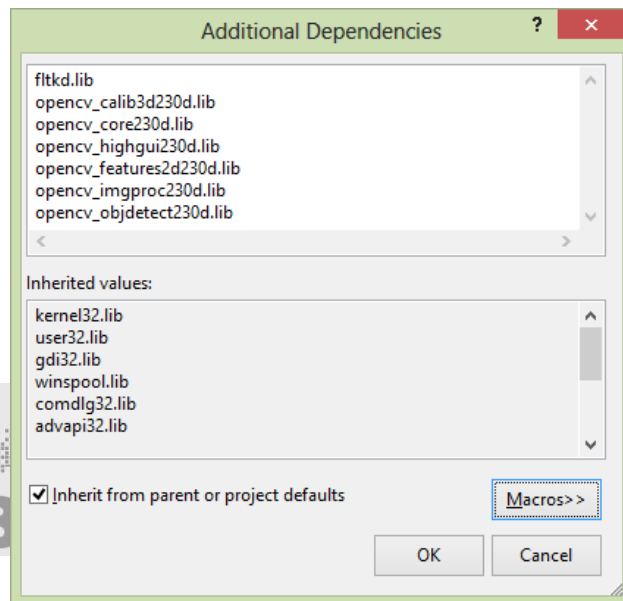


Masih pada **Configuration Properties**, pada bagian kiri, pilih **Linker > Input**. Pada **Additional Dependencies**, klik **Edit**.



Pada dialog **Additional Dependencies**, masukkan nama file berekstensi **.lib** yang dibutuhkan. Dalam pembelajaran ini, dibutuhkan 7 buah **lib** yaitu:

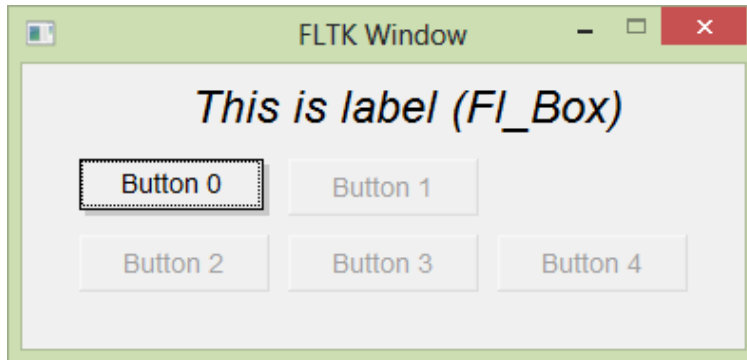
- **fltkd.lib**
- **opencv_calib3d230d.lib**
- **opencv_core230d.lib**
- **opencv_highgui230d.lib**
- **opencv_features2d230d.lib**
- **opencv_imgproc230d.lib**
- **opencv_objdetect230d.lib**



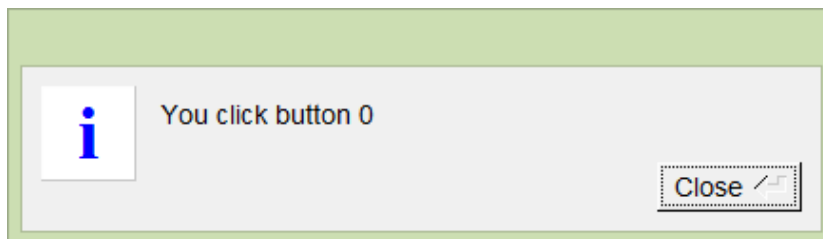
Tambahkan nama **lib** yang lain sesuai dengan kebutuhan. Nama **lib** bisa dilihat di folder **C:/FLTK/lib** atau **C:\OpenCV2.3\build\x86\vc10\lib**. Klik **OK** untuk menutup dialog. Kemudian, klik **OK** untuk menutup jendela **Property**. Kini, **OpenCV2.3** dan **FLTK** sudah dikonfigurasi dan dapat digunakan.

1.4. Exercise

Buatlah sebuah **GUI** dengan tampilan seperti di bawah ini:



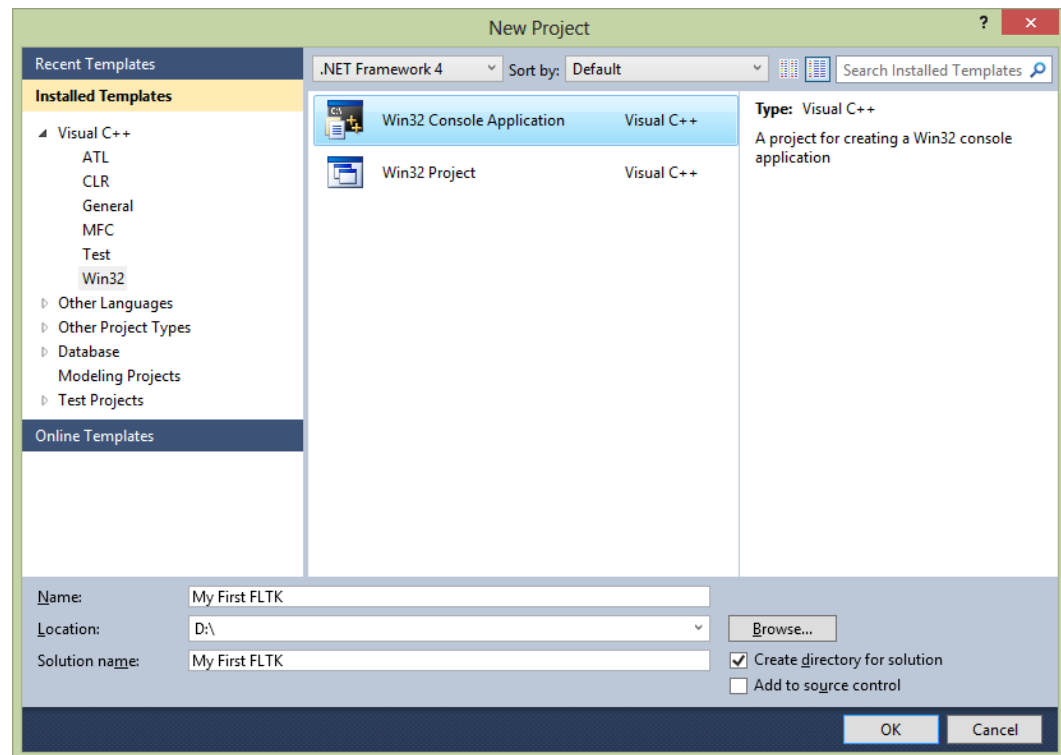
Kemudian, apabila *button* “**Button 0**” ditekan maka akan muncul pesan seperti berikut:



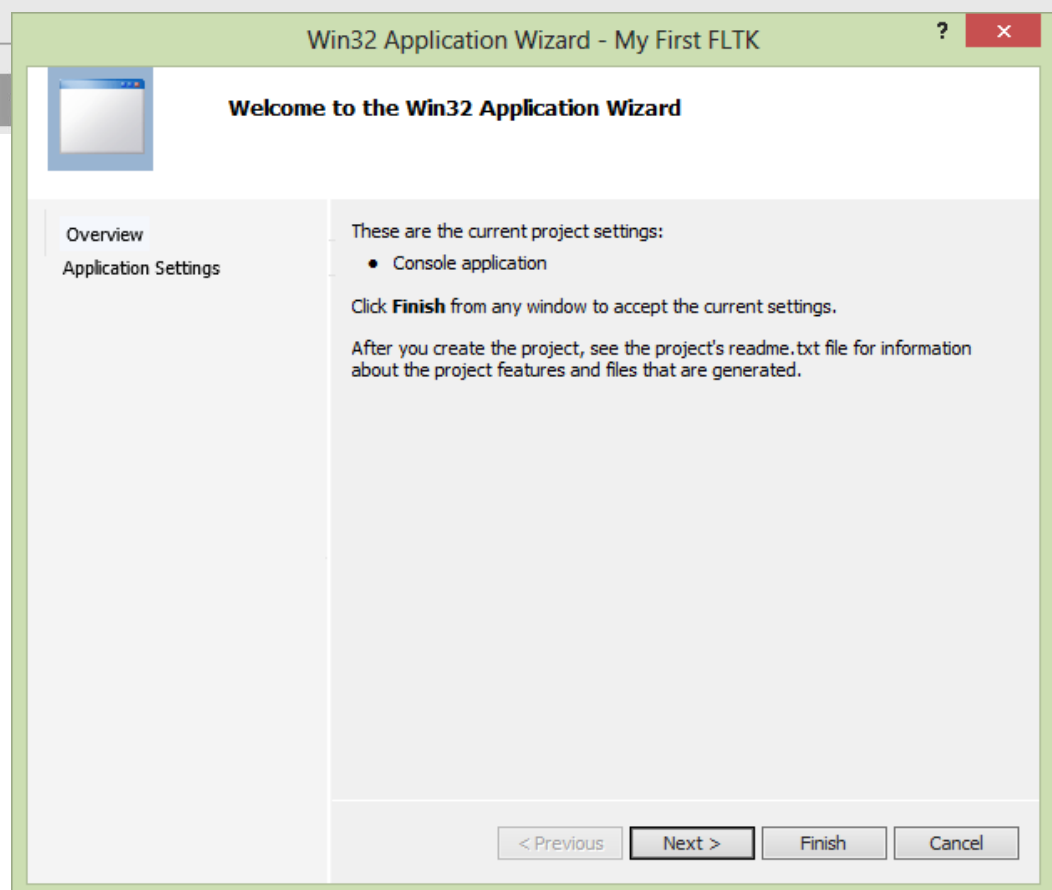
Jawaban:

Langkah-langkah pengerjaan:

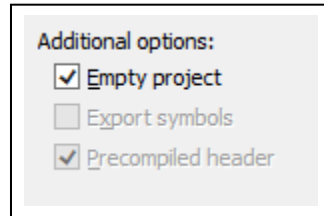
1. Buat sebuah *project* dengan menggunakan **Microsoft Visual Studio 2010**.
 - Jalankan **Microsoft Visual Studio 2010**.
 - Buatlah sebuah *project* dengan cara memilih menu: **File > New > Project**.
 - Pilih **Win32** pada **Visual C++** (pada kolom “**Project types:**”), kemudian pilih “**Win32 Console Application**”. Jangan lupa untuk mengisi nama *project* serta tempat *project* akan dibuat.



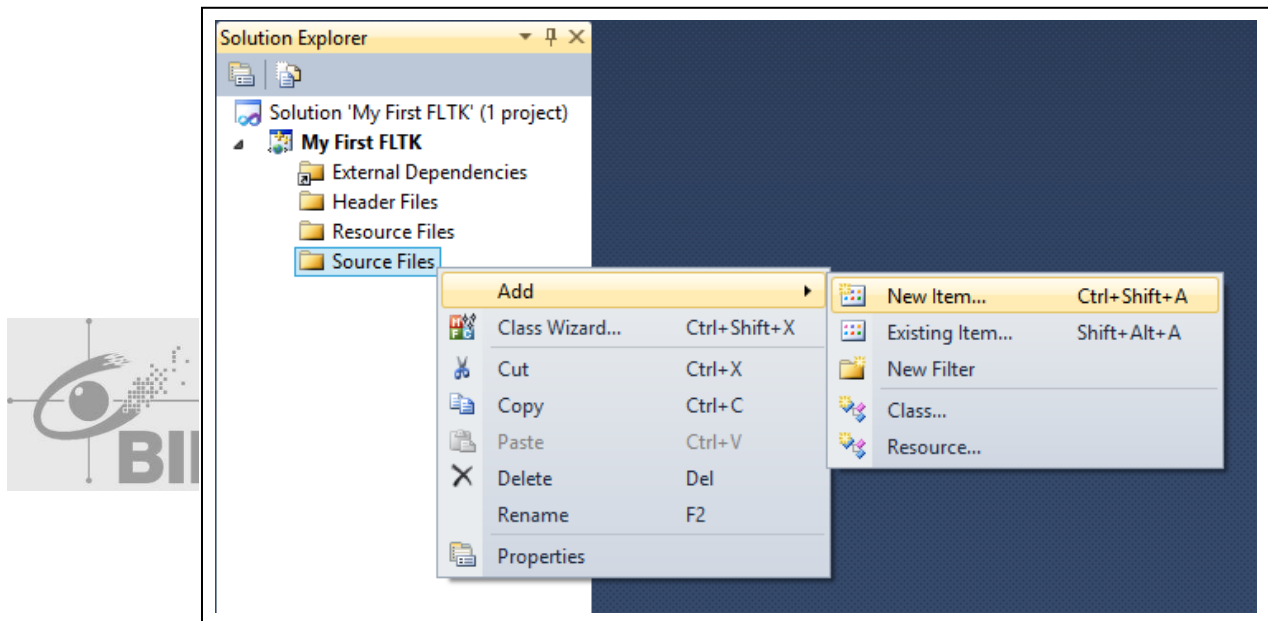
- Setelah itu akan muncul **Windows Application Wizard**. Pilih **Next >**.



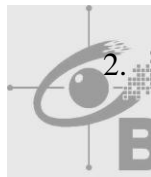
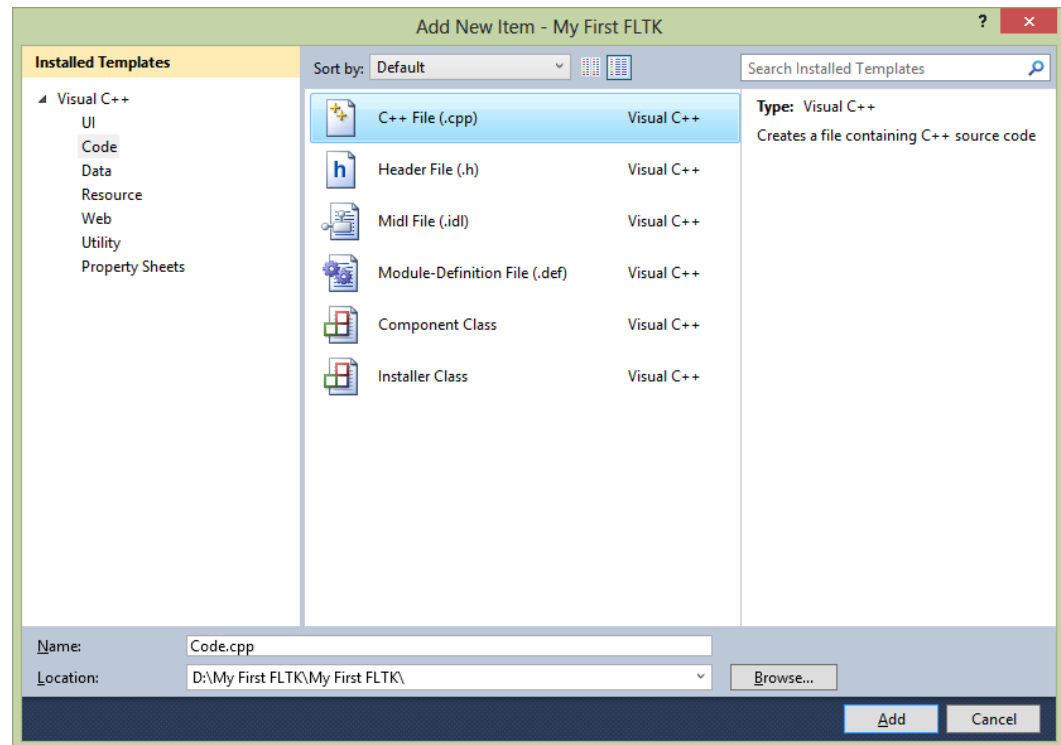
- Kemudian akan muncul *window* **Application Settings**. *Check Empty Project* pada bagian **Additional Options**. Lalu klik **Finish**.



- Buat *file *.cpp* dengan cara memilih *folder Source File*, klik kanan pilih **Add** kemudian **New Item**.



- Kemudian pilih *code* pada **Visual C++** (di bagian kolom **Categories**), dan pilih **C++ File (.cpp)** pada kolom **Templates**. Serta masukkan nama *file code* (pada contoh dibuat **code.cpp**).



2. Ketikkan *include* pada bagian awal *coding*.

```
//Include FLTK
#include <FL\Fl.H>
#include <FL\Fl_Window.H>
#include <FL\Fl_Box.H>
#include <FL\Fl_Button.H>
#include <FL\fl_ask.H>
```

Penjelasan:

Untuk menggunakan **FLTK**, **FL\Fl.H** wajib diikutsertakan di dalam proyek dan sisanya (**FL\Fl_Window.H**, **FL\Fl_Box.H**, **FL\Fl_Button.H**, **FL\fl_ask.H**) adalah komponen yang digunakan dalam membuat tampilan seperti pada soal latihan. **FL\Fl_Window.H** digunakan untuk meng-*import* class **Fl_Window** yang difungsikan untuk membuat *window* baru/tampilan GUI. **FL\Fl_Box.H** digunakan untuk meng-*import* **Fl_Box**, untuk membuat *Label* pada *window*. Sedangkan **FL\Fl_Button.H** digunakan untuk meng-*import* class **Fl_Button**, untuk membuat *button* pada *window*. Dan **FL\fl_ask.H** digunakan untuk meng-*import* class **fl_ask**,

untuk membuat *messagebox* pada *window* ketika terjadi *event*. Ciri-ciri dari **FLTK** adalah semua *class* akan diawali oleh kata “**Fl_**”.

3. Buat dahulu *object-object* dari komponen yang akan digunakan.

```
//Declare UI component
Fl_Window *window;
Fl_Button *button[5];
Fl_Box *label;
```

Penjelasan:

***window** adalah *pointer object* yang dibuat dari *class Fl_Window*, *object* ini digunakan untuk menampilkan *window*.

***button[5]** adalah *pointer object* ini digunakan untuk membuat *button* (jumlah dari *button* adalah lima buah). *Pointer object* ini bersifat *global*, agar semua fungsi dapat memanggil *object-object* tersebut.

***label** adalah *pointer object* yang dibuat dari *class Fl_Box*, *object* ini digunakan untuk membuat *object Box/Label*.

4. Buat **int main** terlebih dahulu. Dimana di dalam fungsi *main*, berisi deklarasi *object-object* yang akan dimunculkan.

```
int main(int argc, char **argv)
{
    //Create the widget
    window = new Fl_Window(380, 150, "FLTK Window");
    label = new Fl_Box(3,3, 400, 40, "This is label (Fl_Box)");
    button[0] = new Fl_Button(30, 50, 100, 30, "Button 0");
    button[1] = new Fl_Button(140, 50, 100, 30, "Button 1");
    button[2] = new Fl_Button(30, 90, 100, 30, "Button 2");
    button[3] = new Fl_Button(140, 90, 100, 30, "Button 3");
    button[4] = new Fl_Button(250, 90, 100, 30, "Button 4");

    //Deactive button 1-4
    for(int i=1;i<5;i++)
    {
        button[i]->deactivate();
    }
}
```

```

//Set callback to button 0
button[0]->callback(cb_button_0);

button[0]->box(FL_SHADOW_BOX);
button[0]->down_color(FL_GREEN);

label->labelsize(24);
label->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

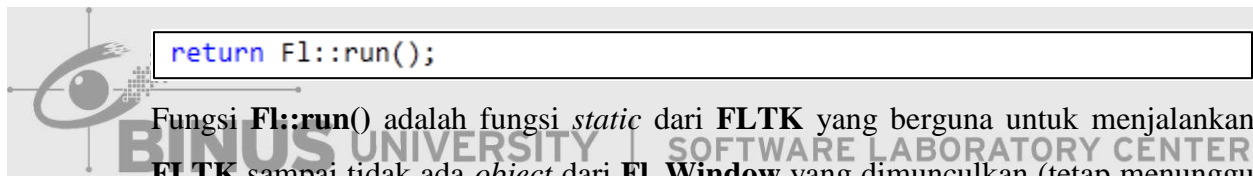
window->end();
window->show(argc,argv);

return Fl::run();
}

```

Penjelasan:

Di dalam fungsi **int main**, harus melakukan pengembalian nilai. Dimana nilai tersebut berasal dari fungsi **Fl::run()**.



Fungsi **Fl::run()** adalah fungsi *static* dari **FLTK** yang berguna untuk menjalankan **FLTK** sampai tidak ada *object* dari **Fl_Window** yang dimunculkan (tetap menunggu sampai semua **FLTK window** ditutup). Apabila tidak ada *window* yang dimunculkan, maka akan dikembalikan nilai 0.

Hal ini sama artinya dengan:

```

while(Fl::wait());
return 0;

```

Dimana **Fl::wait()** adalah fungsi *static* yang berguna untuk mengecek apakah ada *window* dari **FLTK** yang masih aktif.

```
//Create the widget
window = new Fl_Window(380, 150, "FLTK Window");
label = new Fl_Box(3,3, 400, 40, "This is label (Fl_Box)");
button[0] = new Fl_Button(30, 50, 100, 30, "Button 0");
button[1] = new Fl_Button(140, 50, 100, 30, "Button 1");
button[2] = new Fl_Button(30, 90, 100, 30, "Button 2");
button[3] = new Fl_Button(140, 90, 100, 30, "Button 3");
button[4] = new Fl_Button(250, 90, 100, 30, "Button 4");
```

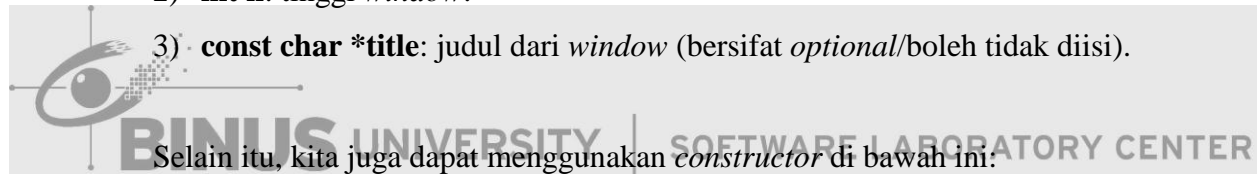
Baris *coding* di atas difungsikan untuk membentuk *object* dari masing-masing *pointer* yang telah dideklarasikan secara *global* sebelumnya.

```
window = new Fl_Window(380, 150, "FLTK Window");
```

Fl_Window::Fl_Window(int w, int h, const char *title = (const char *)0)

Fl_Window memiliki beberapa parameter di-*constructor*-nya, yakni:

- 1) **int w**: lebar *window*.
- 2) **int h**: tinggi *window*.
- 3) **const char *title**: judul dari *window* (bersifat *optional*/boleh tidak diisi).



Selain itu, kita juga dapat menggunakan *constructor* di bawah ini:

Fl_Window::Fl_Window(int x, int y, int w, int h, const char *title = 0)

Dimana:

- 1) **int x**: posisi *window* berdasarkan sumbu x layar (satuan berdasarkan pixel).
- 2) **int y**: posisi *window* berdasarkan sumbu y layar.
- 3) **int w**: lebar *window*.
- 4) **int h**: tinggi *window*.
- 5) **const char *title**: judul dari *window* (bersifat *optional*/boleh tidak diisi).

```
label = new Fl_Box(3,3, 400, 40, "This is label (Fl_Box)");
```

Fl_Box::Fl_Box(int x, int y, int w, int h, const char *I = (const char *)0)

Fl_Box memiliki beberapa parameter di-*constructor*-nya (parameter dari *constructor*

Fl_Box sama dengan parameter *constructor* dari class **Fl_Button**), yakni:

- 1) **int x**: posisi *box* berdasarkan sumbu x *window* **FLTK** (satuan berdasarkan pixel).

- 2) **int y**: posisi *box* berdasarkan sumbu y *window* **FLTK**.
- 3) **int w**: lebar *box*.
- 4) **int h**: tinggi *box*.
- 5) **const char *I**: label/tulisan dari *box* (bersifat *optional*/boleh tidak diisi).

Untuk penjelasan dari masing-masing class dari **FLTK** silahkan dilihat di alamat:

<http://www.fltk.org/doc-1.3/>

```
//Deactive button 1-4
for(int i=1;i<5;i++)
{
    button[i]->deactivate();
}
```

Baris *coding* di atas, difungsikan untuk me-*non*-aktifkan *object-object* dari *button* (fungsi **deactivate()** dijalankan untuk *object button* dari *index* ke-1 sampai *index* ke-4). *Button* yang tidak aktif karena fungsi **deactivate** dapat diaktifkan kembali dengan fungsi **activate**.



```
label->labelsize(24);
label->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);
```

Baris *coding* di atas difungsikan untuk pengaturan *object*.

Pemanggilan fungsi **labelsize()**, difungsikan untuk melakukan pengaturan ukuran tulisan *label* dari *box*.

Sedangkan pemanggilan fungsi **labelfont()**, difungsikan untuk melakukan pengaturan tipe/jenis/efek terhadap tulisan *label* dari *box* dengan menggunakan **Fl_Labeltype**.

```
window->end();
window->show(argc,argv);
```

Fungsi **window->end()** berguna untuk mengakhiri penambahan *widget* (komponen) dari *window*. Penambahan komponen dari *window* bersifat **parent and child** (semua komponen akan menjadi anak dari *window*, sedangkan *window* akan menjadi induk dari semua komponen yang telah didefinisikan sebelumnya).

Fungsi ini harus bersifat eksplisit/dituliskan pada *code* yang dibuat. Berbeda dengan fungsi **begin()** yang secara implisit akan langsung dijalankan ketika semua *object* telah dideklarasikan.

Berikut ini adalah penggunaan dari fungsi **begin()** class **Fl_Window** secara eksplisit:

```

window = new Fl_Window(380, 150, "FLTK Window");
window->begin();
//Widget creation here will be added to window
window->end();
//Widget creation here won't be added to window
window->show(argc, argv);

```

Untuk **window->show(argc, argv)** difungsikan untuk memunculkan *window* yang telah dibuat.

```

//Set callback to button 0
button[0]->callback(cb_button_0);

```

Fungsi **callback()** digunakan untuk menentukan apa yang akan dilakukan apabila terjadi *event*/kejadian pada komponen-komponen tertentu. Pada contoh di atas **button[0]** akan melakukan *callback* terhadap fungsi **cb_button_0()**, dimana artinya apabila **button[0]** ditekan maka akan memanggil fungsi **cb_button_0()**.

5. Buatlah sebuah fungsi **cb_button_0()**.

```

void cb_button_0(Fl_Widget *w, void *v)
{
    fl_message("You click button 0");
}

```

Penjelasan:


Fungsi **cb_button_0(Fl_Widget* w, void* v)** adalah fungsi yang dibuat untuk menangani *event* tertentu (untuk *callback*).

Fungsi untuk **callback()** tersebut harus memiliki parameter berupa **Fl_Widget*** yang berfungsi untuk menangkap komponen pemanggil fungsi tersebut.

Contoh pada kasus di atas, yaitu: **button[0]** merupakan **Fl_Widget** yang melakukan *callback* terhadap fungsi **cb_button_0()**. Maka secara otomatis *object* dari **Fl_Widget*** akan berisi *object* dari **button[0]** (tanpa perlu melakukan penulisan *code* secara eksplisit).

Sedangkan untuk parameter kedua adalah tempat penyimpanan data yang dapat digunakan program pemanggil untuk menyimpan data tertentu. Kita tidak menggunakan parameter kedua ini. Oleh karena itu, kita dapat mempersingkat parameter fungsi dengan tanpa perlu menuliskan nama variabel/*object* pada parameter **void*** (fungsi dapat didefinisikan dengan parameter **cb_button_0(Fl_Widget* w, void*)** tanpa nama variabel pada **void***).

Berikut ini adalah keseluruhan *code* dari *code* di atas:



```
//Include FLTK
#include <FL\Fl.H>
#include <FL\Fl_Window.H>
#include <FL\Fl_Box.H>
#include <FL\Fl_Button.H>
#include <FL\fl_ask.H>

//Declare UI component
Fl_Window *window;
Fl_Button *button[5];
Fl_Box *label;

void cb_button_0(Fl_Widget *w, void *v)
{
    fl_message("You click button 0");
}

int main(int argc, char **argv)
{
    //Create the widget
    window = new Fl_Window(380, 150, "FLTK Window");
    label = new Fl_Box(3,3, 400, 40, "This is label (Fl_Box)");
    button[0] = new Fl_Button(30, 50, 100, 30, "Button 0");
    button[1] = new Fl_Button(140, 50, 100, 30, "Button 1");
    button[2] = new Fl_Button(30, 90, 100, 30, "Button 2");
    button[3] = new Fl_Button(140, 90, 100, 30, "Button 3");
    button[4] = new Fl_Button(250, 90, 100, 30, "Button 4");
```

```
//Deactive button 1-4
for(int i=1;i<5;i++)
{
    button[i]->deactivate();
}

//Set callback to button 0
button[0]->callback(cb_button_0);

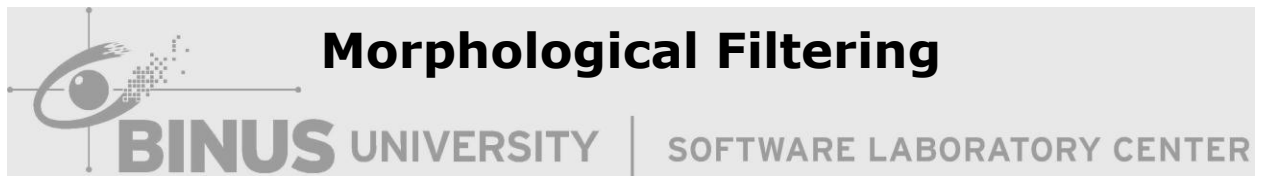
button[0]->box(FL_SHADOW_BOX);
button[0]->down_color(FL_GREEN);

label->labelsize(24);
label->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

window->end();
window->show(argc,argv);

return Fl::run();
}
```


Chapter 02



2.1 Smoothing

Smoothing (disebut juga *blur*) adalah operasi pengolahan citra sederhana yang sering digunakan. Ada banyak alasan untuk menghaluskan, tetapi biasanya dilakukan untuk mengurangi *noise*, gangguan pada image. *Smoothing* juga penting ketika kita ingin mengurangi resolusi dari suatu gambar.

2.2 Grayscale

Dalam komputasi, suatu citra digital *grayscale* atau *greyscale* adalah suatu citra dimana nilai dari setiap *pixel* merupakan *sample* tunggal. Citra yang ditampilkan terdiri atas warna “**abu-abu**”, bervariasi di warna hitam pada bagian yang intensitas terlemah dan warna putih pada intensitas terkuat. Citra *grayscale* berbeda dengan citra “hitam-putih”. Dimana pada konteks komputer, citra hitam putih hanya terdiri atas 2 warna saja yaitu “**hitam**” dan “**putih**” saja.

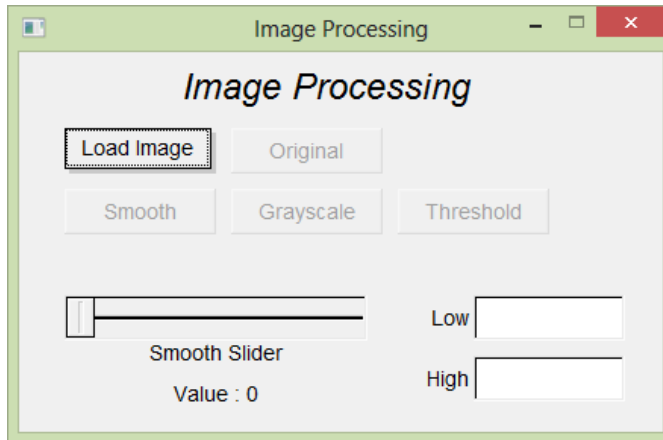
Pada citra *grayscale* warna bervariasi antara hitam dan putih, tetapi variasi warna diantaranya sangat banyak. Citra *grayscale* seringkali merupakan perhitungan dari intensitas cahaya pada setiap *pixel* pada spektrum elektromagnetik *single band*. Citra *grayscale* disimpan dalam format 8 bit untuk setiap *sample pixel*, yang memungkinkan sebanyak 256 intensitas. Format ini sangat membantu dalam pemrograman karena manipulasi bit yang tidak terlalu banyak. Pada aplikasi lain seperti pada aplikasi *medical imaging* dan *remote sensing* biasa juga digunakan format 10, 12 maupun 16 bit.

2.3 Threshold

Threshold merupakan suatu operasi untuk mengubah gambar *gray-scale* ataupun gambar berwarna menjadi gambar biner berdasarkan pada nilai *threshold* tertentu. Adapun gambar biner merupakan gambar yang hanya memiliki dua nilai intensitas warna untuk setiap *pixel*-nya, yaitu nilai **255 (warna putih)** dan **0 (warna hitam)**.

2.4. Exercise

Buatlah sebuah program untuk melakukan operasi *smoothing/blur*, *grayscale*, *threshold*, dan *convert color*. Seperti berikut ini:



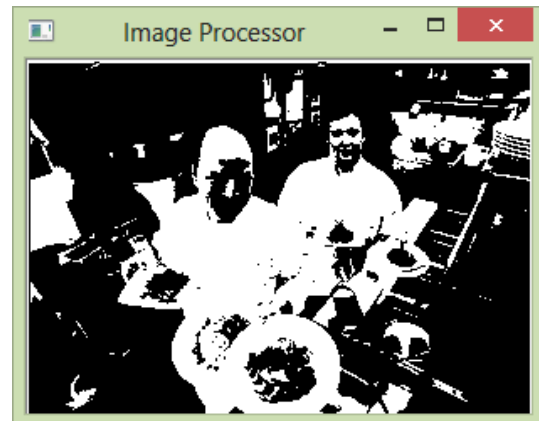
Hasil dari **Smoothing**:



Hasil dari **Grayscale/Convert Color**:



Hasil dari **Threshold** (min:128, max:255):



Jawaban:

Langkah-langkah pengerjaan:

1. Buat *project* baru dari **Microsoft Visual Studio 2010 C++**. Cara membuat *project* baru masih sama seperti yang telah kita pelajari pada **bab I**.

2. Tuliskan semua *include* file yang dibutuhkan.

```
//Include FLTK
#include<FL\Fl.H>
#include<FL\Fl_Window.H>
#include<FL\Fl_Box.H>
#include<FL\Fl_Button.H>
#include<FL\fl_ask.H>
#include<FL\Fl_Input.H>
#include<FL\Fl_Slider.H>
#include<FL\Fl_File_Chooser.H>

//Include OpenCV
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\core\core.hpp>
#include<opencv2\imgproc\imgproc.hpp>

using namespace cv;
```

Penjelasan:

Untuk *include* file **FLTK**, diperlukan beberapa komponen seperti **Window**, **Box**, **Button**, **Ask** (*messagebox* **FLTK** untuk meminta inputan dari user), **Input** (*text input*

dari **FLTK**), **Slider** (*slider* dari **FLTK**), **File Chooser** (*window* untuk mengambil *file* dari suatu *directory*).

Sedangkan untuk *include file* **OpenCV** dibutuhkan file **core.hpp**, **highgui.hpp**, serta **imgproc.hpp**.

3. Buat semua *object* yang akan digunakan.

```
//Declare UI component
Fl_Window *window;
Fl_Button *button[5];
Fl_Box *label, *value_text;
Fl_Button *nbutton;
Fl_File_Chooser *chooser;
Fl_Slider *slider;
Fl_Input *low, *high;

//Declare OpenCV image object
Mat original, edited;
```



Penjelasan:

Mat adalah *class* yang menyimpan data gambar dalam bentuk matriks.

BINUS UNIVERSITY
SOFTWARE LABORATORY CENTER

4. Di dalam **int main**, buatlah baris *coding* seperti berikut ini:

```
//Create the widget
window = new Fl_Window(425, 250, "Image Processing");
label = new Fl_Box(3, 3, 400, 40, "Image Processing");
button[0] = new Fl_Button(30, 50, 100, 30, "Load Image");
button[1] = new Fl_Button(140, 50, 100, 30, "Original");
button[2] = new Fl_Button(30, 90, 100, 30, "Smooth");
button[3] = new Fl_Button(140, 90, 100, 30, "Grayscale");
button[4] = new Fl_Button(250, 90, 100, 30, "Threshold");
slider = new Fl_Slider(30, 160, 200, 30, "Smooth Slider");
value_text = new Fl_Box(30, 200, 200, 50, "Value : 0");
low = new Fl_Input(300, 160, 100, 30, "Low");
high = new Fl_Input(300, 200, 100, 30, "High");
chooser = new Fl_File_Chooser("../", "Image File(*.{jpg,png})",
    Fl_File_Chooser::SINGLE, "Choose Image");
```

Penjelasan:

```
window = new Fl_Window(425, 250, "Image Processing");
```


Digunakan untuk membuat *window* yang akan dimunculkan.

```
label = new Fl_Box(3, 3, 400, 40, "Image Processing");
```

Digunakan untuk memuat **label/box**, yang digunakan untuk menampilkan tulisan pada *window*.

```
button[0] = new Fl_Button(30, 50, 100, 30, "Load Image");
button[1] = new Fl_Button(140, 50, 100, 30, "Original");
button[2] = new Fl_Button(30, 90, 100, 30, "Smooth");
button[3] = new Fl_Button(140, 90, 100, 30, "Grayscale");
button[4] = new Fl_Button(250, 90, 100, 30, "Threshold");
```

Digunakan untuk membuat *button* yang akan digunakan untuk menentukan masing-masing aksi pada *window*.



```
slider = new Fl_Slider(30, 160, 200, 30, "Smooth Slider");
```

Digunakan untuk membuat *slider*, parameter pada *slider* sama seperti parameter pada **Fl_Text**.

```
value_text = new Fl_Box(30, 200, 200, 50, "Value : 0");
```

value_text akan digunakan untuk menampilkan *value* dari *slider* yang ada.

```
low = new Fl_Input(300, 160, 100, 30, "Low");
high = new Fl_Input(300, 200, 100, 30, "High");
```

Fl_Input akan digunakan untuk menyimpan inputan *user* berupa *text*. Parameter dan *constructor* yang digunakan kurang lebih sama dengan *box* atau *button*, yaitu:

- 1) **int x**: posisi *input* berdasarkan sumbu x *window FLTK* (satuan berdasarkan *pixel*).
- 2) **int y**: posisi *input* berdasarkan sumbu y *window FLTK*.
- 3) **int w**: lebar *input*.
- 4) **int h**: tinggi *input*.

- 5) **const char *title:** *label/tulisan dari input* (bersifat *optional*/boleh tidak diisi).

```
chooser = new Fl_File_Chooser("../", "Image File(*.{jpg,png})",
    Fl_File_Chooser::SINGLE, "Choose Image");
```

Digunakan untuk membuat *window file chooser*. Memiliki parameter pada *constructor*, berupa:

- 1) **Path:** tempat pertama kali **file chooser** akan membuka *file*.
- 2) **Pattern:** ekstensi *file* apa saja yang akan menjadi *default* ekstensi dari **Fl_File_chooser**. Untuk menambah ekstensi, dapat digunakan **simbol |**, contoh apabila kita akan menambah tipe video, maka pada parameter kedua dapat diberikan “**Image File(*.{jpg,png})|Video File(*.{avi,mp4})**”.
- 3) **Enumerasi tipe file chooser:** terdiri dari beberapa enumerasi, antara lain:
 - **SINGLE:** memperbolehkan *user* untuk membuka *single file*.
 - **MULTI:** memperbolehkan *user* untuk membuka satu atau banyak *file*.
 - **CREATE:** memperbolehkan *user* untuk memilih *file*, satu yang sudah ada atau menentukan nama *file* baru.
 - **DIRECTORY:** memperbolehkan *user* untuk memilih salah satu *directory* yang ada.
- 4) **Title:** judul dari *window file chooser*.

Lalu buatlah baris *coding* seperti di bawah ini sebagai lanjutan dari baris *coding* sebelumnya:

```
//Deactive button 1-4
for(int i=1;i<5;i++)
{
    button[i]->deactivate();
}

//Set callback to button
button[0]->callback(load_image);
button[1]->callback(original_image);
button[2]->callback(smooth_image);
button[3]->callback(grayscale_image);
button[4]->callback(threshold_image);
```

```

button[0]->box(FL_SHADOW_BOX);
button[0]->down_color(FL_GREEN);

//Set callback to slider
slider->callback(change_text);

slider->minimum(0);
slider->maximum(100);
slider->step(1);
slider->type(FL_HOR_NICE_SLIDER);
slider->slider_size(0.001);

label->labelsize(24);
label->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

window->end();
window->show(argc,argv);
return Fl::run();

```

Penjelasan:



```

//Deactive button 1-4
for(int i=1;i<5;i++)
{
    button[i]->deactivate();
}

```

Membuat semua *button* menjadi *non-aktif* kecuali *button load image*.

```

//Set callback to button
button[0]->callback(load_image);
button[1]->callback(original_image);
button[2]->callback(smooth_image);
button[3]->callback(grayscale_image);
button[4]->callback(threshold_image);

```

Menentukan *callback* dari semua *button*. Fungsi yang akan digunakan harus sudah didefinisikan terlebih dahulu.


```
button[0]->box(FL_SHADOW_BOX);
button[0]->down_color(FL_GREEN);
```

Menentukan efek yang akan diberikan kepada *button* ke-0.

button[0]->box(FL_SHADOW_BOX) digunakan untuk memberikan efek *shadow* kepada **button** ke-0.

button[0]->down_color(FL_GREEN) digunakan untuk memberikan efek warna hijau kepada **button** ke-0 ketika ditekan.

```
//Set callback to slider
slider->callback(change_text);

slider->minimum(0);
slider->maximum(100);
slider->step(1);
slider->type(FL_HOR_NICE_SLIDER);
slider->slider_size(0.001);
```

Digunakan untuk memberikan pengaturan kepada *slider*.

slider->callback(change_text) digunakan untuk menentukan *callback* dari **slider**.

slider->maximum(100) digunakan untuk menentukan nilai maksimum dari **slider**.

slider->minimum(0) digunakan untuk menentukan nilai minimal dari *slider*.

slider->step(1) digunakan untuk menentukan selisih jarak yang dapat dilewati oleh **slider**.

slider->type(FL_HOR_FILL_SLIDER) digunakan untuk menentukan tipe dari **slider**. Memiliki tetapan/enumerasi antara lain:

- **FL_VERTICAL**: untuk membuat *slider* dengan tipe *vertical* (*default* dari enumerasi *slider*).
- **FL_HORIZONTAL**: untuk membuat tipe *slider* dengan tipe *horizontal*.
- **FL_VERT_FILL_SLIDER**: untuk membuat *slider* dengan tipe vertikal dan terisi.
- **FL_HOR_FILL_SLIDER**: untuk membuat *slider* dengan tipe *horizontal* dan terisi.
- **FL_VERT_NICE_SLIDER**: untuk membuat *slider* dengan tipe vertikal dan memiliki tampilan yang baik.

- **FL_HOR_NICE_SLIDER**: untuk membuat *slider* dengan tipe *horizontal* dan memiliki tampilan yang baik.

slider->**slider_size(0.001)** untuk menentukan ukuran ketebalan *scroll*.

```
label->labelsize(24);
label->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

window->end();
window->show(argc,argv);
return Fl::run();
```

Digunakan untuk pengaturan komponen dan memunculkan *window*.

5. Membuat fungsi **load_image()** untuk di-*callback* oleh **button[0]**.



```
int isImageFile(const char *fileName)
{
    int len = strlen(fileName);

    if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'j' &&
        tolower(fileName[len-2])== 'p' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }else if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'p' &&
        tolower(fileName[len-2])== 'n' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }
    else return 0;
}

void load_image(Fl_Widget *w, void *v)
{
    chooser->show();
    while(chooser->shown())
    {
        Fl::wait();
    }
    if(chooser->value()!=0 && isImageFile(chooser->value()))
    {
```

```

        original = imread(chooser->value(1));
        imshow("Image Processor",original);

        for(int i=0;i<5;i++)
        {
            button[i]->activate();
        }
    }
}

```

Penjelasan:

Fungsi `int isImageFile(const char *fileName)`, digunakan untuk melakukan pengecekan terhadap format (harus **.jpg** dan **.png**).

Sedangkan pada fungsi `load_image`, digunakan untuk memanggil **file chooser** lalu memunculkannya di *window* baru.

```
chooser->show();
```



Digunakan untuk memunculkan **file chooser** yang telah dibuat sebelumnya.

```

while(chooser->shown())
{
    Fl::wait();
}

```

Digunakan untuk membuat seluruh proses dari **FLTK** menunggu **selama file chooser** masih terbuka.

```

if(chooser->value()!=0 && isImageFile(chooser->value()))
{
    original = imread(chooser->value(1));
    imshow("Image Processor",original);

    for(int i=0;i<5;i++)
    {
        button[i]->activate();
    }
}

```

Untuk melakukan pengecekan apakah *file* telah dipilih atau belum serta melakukan pengecekan format dengan cara memanggil fungsi **isImageFile()**.

Apabila berhasil melewati **if/pengecekan**, maka program akan mengeksekusi baris selanjutnya.

chooser->value() digunakan untuk mendapatkan nilai dari *file* yang telah dipilih.

Fungsi **imread()** digunakan untuk me-load *image*. Hasil *image* yang berhasil di-load tersebut akan ditampung ke dalam variabel dari *class Mat* dengan nama **original**.

```
imshow("Image Processor",original);
```

Fungsi **imshow()** digunakan untuk memunculkan *image* ke dalam *window* yang langsung dibuat ketika fungsi ini dijalankan.

```
for(int i=0;i<5;i++)
{
    button[i]->activate();
}
```

Digunakan untuk mengaktifkan seluruh **button** yang ada.

6. Membuat fungsi **original_image()** untuk di-*callback* oleh **button[1]**.

```
void original_image(Fl_Widget *w, void *v)
{
    imshow("Image Processor",original);
}
```

Penjelasan:

Fungsi ini digunakan untuk mengembalikan gambar ke gambar semula (gambar ketika pertama kali di-load).

7. Membuat fungsi **smooth_image()** dan **change_text()** untuk di-*callback* oleh **button[2]** dan **slider**.

```

void smooth_image(Fl_Widget *w, void *v)
{
    int val = slider->value();
    if(val%2!=1)
        val+=1;
    blur(original, edited, Point(val,val));
    imshow("Image Processor",edited);
}

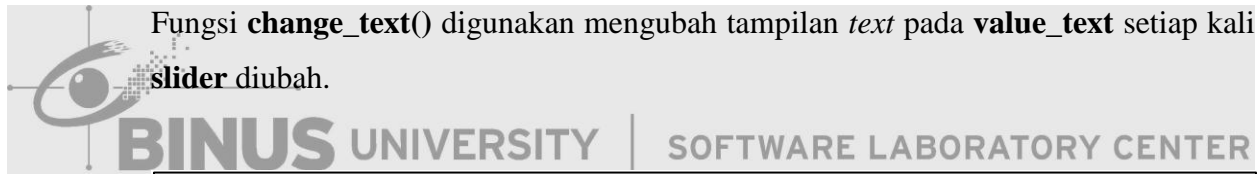
void change_text(Fl_Widget *w, void *v)
{
    char val[100];
    sprintf(val,"Value : %.0f",slider->value());
    value_text->label(val);
}

```

Penjelasan:

Fungsi **smooth_image()** digunakan untuk mengubah/mem-*blur*-kan gambar sesuai dengan *value* yang dipilih pada **slider**.

Fungsi **change_text()** digunakan mengubah tampilan *text* pada **value_text** setiap kali **slider** diubah.



```

int val = slider->value();
if(val%2!=1)
    val+=1;

```

Baris *coding* di atas digunakan untuk menentukan nilai dari besaran **blur** yang ingin diberikan. Nilai parameter *blur* yang akan digunakan didapat dari *value* **slider** yang kemudian akan disimpan dalam variabel **val**. *Blur* hanya bisa menerima angka ganjil, maka **value** dari **val** perlu kita ubah menjadi bilangan ganjil.

```

blur(original, edited, Point(val,val));

```

Fungsi **blur()** digunakan untuk melakukan *smoothing* terhadap gambar.

Memiliki parameter sebagai berikut:

- 1) **Source:** gambar awal untuk di-*smooth*.
- 2) **Destination:** gambar tujuan untuk menampung hasil *smoothing*.
- 3) **Size:** nilai *smoothing* dalam tipe Point (harus bernilai positif dan ganjil).

```
imshow("Image Processor",edited);
```

Untuk memunculkan *image* hasil proses *smoothing*.

8. Membuat fungsi **grayscale_image()** untuk di-*callback* oleh **button[3]**.

```
void grayscale_image(Fl_Widget *w, void *v)
{
    cvtColor(original,edited,CV_RGB2GRAY);
    imshow("Image Processor",edited);
}
```

Penjelasan:

Fungsi ini digunakan untuk memunculkan gambar **grayscale** dari gambar yang *original*. Gambar yang *original* tidak diubah. Sebagai gantinya, diperlukan alokasi memori tambahan untuk menampung hasil gambar yang telah diubah (di-*grayscale*).

```
cvtColor(original,edited,CV_RGB2GRAY);
```

Baris *coding* di atas menggunakan fungsi dari **cvtColor()** untuk melakukan konversi warna gambar.

Memiliki parameter, antara lain:

- 1) **Source:** *image* awal yang akan dikonversi warnanya (variabel awal dengan tipe data *class Mat*).
- 2) **Destination:** target *image* hasil konversi (variabel tujuan dengan tipe data *class Mat*).
- 3) **Color Conversion Code:** kode konversi *image*.

Table tetapan dari **Color Conversion Code**:

CV_BGR2BGRA	Convert BGR color to BGRA color
CV_BGRA2BGR	Convert BGRA color to BGR color
CV_BGR2GRAY	Convert BGR color to GRAY color
CV_GRAY2BGR	Convert GRAY color to BGR color
CV_GRAY2BGRA	Convert GRAY color to BGRA color
CV_BGRA2GRAY	Convert BGRA color to GRAY color
CV_BGR2BGR565	Convert BGR color to BGR565 color

CV_BGR5652BGR	Convert BGR565 color to BGR color
CV_BGRA2BGR565	Convert BGRA color to BGR565 color
CV_BGR5652BGRA	Convert BGR565 color to BGRA color
CV_GRAY2BGR565	Convert GRAY color to BGR565 color
CV_BGR5652GRAY	Convert BGR565 color to GRAY color
CV_BGR2BGR555	Convert BGR color to BGR555 color
CV_BGR5552BGR	Convert BGR555 color to BGR color
CV_BGRA2BGR555	Convert BGRA color to BGR555 color
CV_BGR5552BGRA	Convert BGR555 color to BGRA color
CV_GRAY2BGR555	Convert GRAY color to BGR555 color
CV_BGR5552GRAY	Convert BGR555 color to GRAY color
CV_BGR2XYZ	Convert BGR color to XYZ color
CV_XYZ2BGR	Convert XYZ color to BGR color
CV_BGR2YCrCb	Convert BGR color to YCrCb color
CV_YCrCb2BGR	Convert YCrCb color to BGR color
CV_BGR2HSV	Convert BGR color to HSV color
CV_BGR2Lab	Convert BGR color to Lab color
CV_BayerBG2BGR	Convert BayerBG color to BGR color
CV_BayerGB2BGR	Convert BayerGB color to BGR color
CV_BayerRG2BGR	Convert BayerRG color to BGR color
CV_BayerGR2BGR	Convert BayerGR color to BGR color
CV_BGR2Luv	Convert BGR color to Luv color
CV_BGR2HLS	Convert BGR color to HLS color
CV_HSV2BGR	Convert HSV color to BGR color
CV_Lab2BGR	Convert Lab color to BGR color
CV_Luv2BGR	Convert Luv color to BGR color
CV_HLS2BGR	Convert HLS color to BGR color

```
imshow("Image Processor",edited);
```

Untuk memunculkan *image* hasil proses konversi warna ke dalam bentuk *grayscale*.

9. Membuat fungsi **threshold_image()** untuk di-*callback* oleh **button[4]**.

```
void threshold_image(Fl_Widget *w, void *v)
{
    double lowNo, HighNo;
    lowNo = atoi(low->value());
    HighNo = atoi(high->value());
    if(lowNo>0 && HighNo>0)
    {
        cvtColor(original,edited,CV_RGB2GRAY);
        threshold(edited,edited,lowNo,HighNo,CV_THRESH_BINARY);
        imshow("Image Processor",edited);
    }
    else
    {
        fl_message("Please insert valid input for threshold");
    }
}
```

Penjelasan:



```
double lowNo, HighNo;
lowNo = atoi(low->value());
HighNo = atoi(high->value());
```

Baris *coding* di atas digunakan untuk membuat variabel dan mengambil hasil inputan *user* di dalam **Fl_Input low** dan **hi**.

```
if(lowNo>0 && HighNo>0)
```

Untuk melakukan pengecekan terhadap nilai inputan *user*. Fungsi **atoi** akan melakukan konversi **string** dari **low->value()** dan **hi->value()** dimana **atoi** akan mengembalikan nilai **nol** apabila parameter **string** tidak dapat dikonversi. Karenanya, kita cukup memvalidasi nilai konversi harus lebih besar dari nol.

```
cvtColor(original,edited,CV_RGB2GRAY);
```

Membuat *image* menjadi *grayscale*.

Catatan: dalam melakukan operasi *threshold*, untuk mendapatkan hasil *threshold* yang lebih baik maka gambar harus dijadikan *grayscale* terlebih dahulu.


```
threshold(edited,edited,lowNo,HighNo,CV_THRESH_BINARY);
```

Fungsi **threshold()** digunakan untuk melakukan *thresholding*. Memiliki parameter, antara lain:

- 1) **Source**: gambar awal yang akan dilakukan *thresholding*.
- 2) **Destination**: gambar tujuan hasil dari *thresholding*.
- 3) **int lowest value**: nilai terendah *thresholding*.
- 4) **int highest value**: nilai tertinggi *thresholding*.
- 5) **int threshold_type**: tipe dari *thresholding*. **Threshold** memiliki beberapa tipe, antara lain:
 - **CV_THRESH_BINARY**, value = value > threshold ? max_value : 0
 - **CV_THRESH_BINARY_INV**, value = value > threshold ? 0 : max_value.
 - **CV_THRESH_TRUNC**, value = value > threshold ? threshold : value.
 - **CV_THRESH_TOZERO**, value = value > threshold ? value : 0.
 - **CV_THRESH_TOZERO_INV**, value = value > threshold ? 0 : value.

Tipe-tipe dari *thresholding* inilah, yang akan digunakan sebagai algoritma dari perhitungan besaran nilai *pixel* dari masing-masing titik pada gambar.

```
imshow("Image Processor",edited);
```

Untuk memunculkan gambar hasil *thresholding* dari source **edited**.

Berikut ini adalah keseluruhan *code* dari *code* di atas:

```
//Include FLTK
#include<FL\Fl.H>
#include<FL\Fl_Window.H>
#include<FL\Fl_Box.H>
#include<FL\Fl_Button.H>
#include<FL\fl_ask.H>
#include<FL\Fl_Input.H>
#include<FL\Fl_Slider.H>
#include<FL\Fl_File_Chooser.H>
```

```

//Include OpenCV
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\core\core.hpp>
#include<opencv2\imgproc\imgproc.hpp>

using namespace cv;

//Declare UI component
Fl_Window *window;
Fl_Button *button[5];
Fl_Box *label, *value_text;
Fl_Button *nbutton;
Fl_File_Chooser *chooser;
Fl_Slider *slider;
Fl_Input *low, *high;

//Declare OpenCV image object
Mat original, edited;

int isImageFile(const char *fileName)
{
    int len = strlen(fileName);

    if(tolower(fileName[len-4])=='.' &&
        tolower(fileName[len-3])=='j' &&
        tolower(fileName[len-2])=='p' &&
        tolower(fileName[len-1])=='g'){
        return 1;
    }else if(tolower(fileName[len-4])=='.' &&
        tolower(fileName[len-3])=='p' &&
        tolower(fileName[len-2])=='n' &&
        tolower(fileName[len-1])=='g'){
        return 1;
    }
    else return 0;
}

void load_image(Fl_Widget *w, void *v)
{
    chooser->show();
    while(chooser->shown())
    {
        Fl::wait();
    }
}

```

```

    if(chooser->value() != 0 && isImageFile(chooser->value()))
    {
        original = imread(chooser->value(1));
        imshow("Image Processor",original);

        for(int i=0;i<5;i++)
        {
            button[i]->activate();
        }
    }
}

void original_image(Fl_Widget *w, void *v)
{
    imshow("Image Processor",original);
}

void smooth_image(Fl_Widget *w, void *v)
{
    int val = slider->value();
    if(val%2!=1)
        val+=1;
    blur(original, edited, Point(val,val));
    imshow("Image Processor",edited);
}

void change_text(Fl_Widget *w, void *v)
{
    char val[100];
    sprintf(val,"Value : %.0f",slider->value());
    value_text->label(val);
}

void grayscale_image(Fl_Widget *w, void *v)
{
    cvtColor(original,edited,CV_RGB2GRAY);
    imshow("Image Processor",edited);
}

void threshold_image(Fl_Widget *w, void *v)
{
    double lowNo, HighNo;
    lowNo = atoi(low->value());
    HighNo = atoi(high->value());
}

```

```

        if(lowNo>0 && HighNo>0)
        {
            cvtColor(original,edited,CV_RGB2GRAY);
            threshold(edited,edited,lowNo,HighNo,CV_THRESH_BINARY);
            imshow("Image Processor",edited);
        }
        else
        {
            fl_message("Please insert valid input for threshold");
        }
    }
}

int main(int argc, char **argv)
{
    //Create the widget
    window = new Fl_Window(425, 250, "Image Processing");
    label = new Fl_Box(3, 3, 400, 40, "Image Processing");
    button[0] = new Fl_Button(30, 50, 100, 30, "Load Image");
    button[1] = new Fl_Button(140, 50, 100, 30, "Original");
    button[2] = new Fl_Button(30, 90, 100, 30, "Smooth");
    button[3] = new Fl_Button(140, 90, 100, 30, "Grayscale");
    button[4] = new Fl_Button(250, 90, 100, 30, "Threshold");
    slider = new Fl_Slider(30, 160, 200, 30, "Smooth Slider");
    value_text = new Fl_Box(30, 200, 200, 50, "Value : 0");
    low = new Fl_Input(300, 160, 100, 30, "Low");
    high = new Fl_Input(300, 200, 100, 30, "High");
    chooser = new Fl_File_Chooser("../", "Image File(*.{jpg,png})",
        Fl_File_Chooser::SINGLE, "Choose Image");

    //Deactive button 1-4
    for(int i=1;i<5;i++)
    {
        button[i]->deactivate();
    }

    //Set callback to button
    button[0]->callback(load_image);
    button[1]->callback(original_image);
    button[2]->callback(smooth_image);
    button[3]->callback(grayscale_image);
    button[4]->callback(threshold_image);
}

```

```
button[0]->box(FL_SHADOW_BOX);
button[0]->down_color(FL_GREEN);

//Set callback to slider
slider->callback(change_text);

slider->minimum(0);
slider->maximum(100);
slider->step(1);
slider->type(FL_HOR_NICE_SLIDER);
slider->slider_size(0.001);

label->labelsize(24);
label->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

window->end();
window->show(argc,argv);
return Fl::run();
}
```

Chapter 03



3.1. Canny Edge Detection

Canny Edge Detection adalah sebuah metode pengenalan garis/tepi yang akan menghasilkan garis-garis *pixel* pada ujung (tepi) dari suatu daerah yang tebal. *Canny Edge Detection Thinning Algorithm* merupakan metode yang melakukan deteksi terhadap ujung (tepi) dari suatu *image*.

Memiliki keunggulan karena:

1. *Smooth image* dengan **Gaussian** mengoptimalkan *trade-off* antara *noise filtering* dan *edge localization*.
2. Menghitung **Gradient Magnitude** menggunakan pendekatan parsial derivatif 2x2 *filter*.
3. *Edges by line* menerapkan penekanan *non-maximal* dengan besarnya *gradient*.
4. Deteksi tepi dengan **thresholding** ganda.

3.2. Sobel Edge Detection

Sobel edge detection adalah salah satu metode dalam *image processing* yang berguna untuk mendeteksi tepi (*edge*) suatu objek dalam gambar digital. *Edge* dapat terjadi karena adanya perubahan atau perbedaan (*gradient*) nilai *pixel* yang cukup berpengaruh antara suatu *pixel* terhadap *pixel* yang berada di sekitarnya.

Secara umum, **sobel edge detection** dibedakan menjadi dua macam, yakni:

- 1) **Sobel horizontal**: dimana pencarian *edge* dilakukan searah sumbu x gambar.
- 2) **Sobel vertikal**: dimana pencarian *edge* dilakukan searah sumbu y gambar.

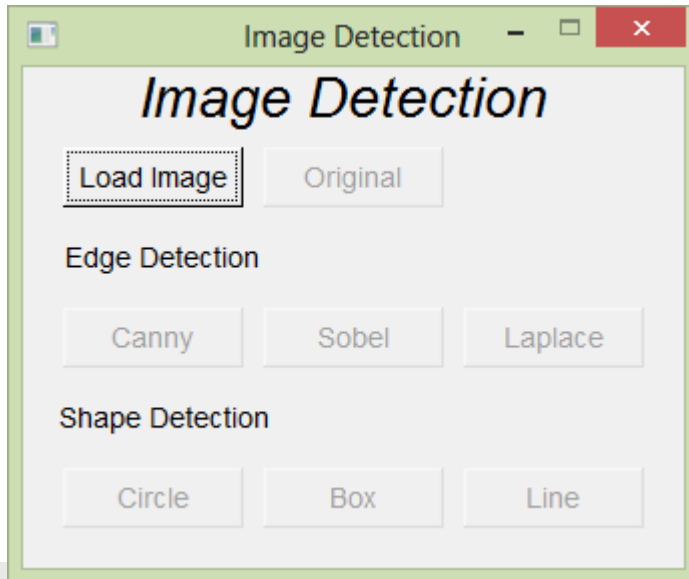
3.3. Laplace Edge Detection

Laplace edge detection menggunakan *matrix Laplace 5x5*. Dimana *matrix Laplace 5x5* yang digunakan adalah *convolution mask* untuk mendekati turunan kedua, tidak seperti metode **Sobel** yang mendekati *gradient*. Dan bukannya 2 *mask*, 3x3 *matrix Sobel*, satu untuk arah x dan y, **Laplace** menggunakan 15x5 *masker* untuk turunan 2 baik diarah x dan y. Namun, karena pelindung yang mendekati ukuran turunan kedua pada gambar, maka mereka sangat peka terhadap *noise*.

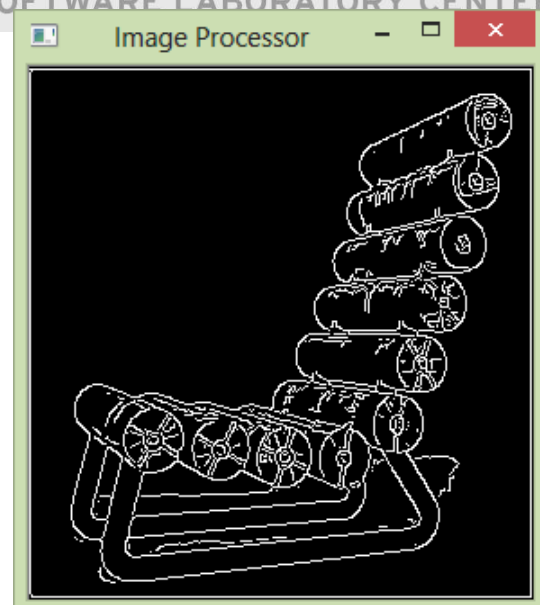
3.4. Exercise

Buatlah sebuah program untuk melakukan operasi **canny edge detection**, **sobel edge detection**, **laplace edge detection**, **circle detection**, **box detection** dan **line detection**.

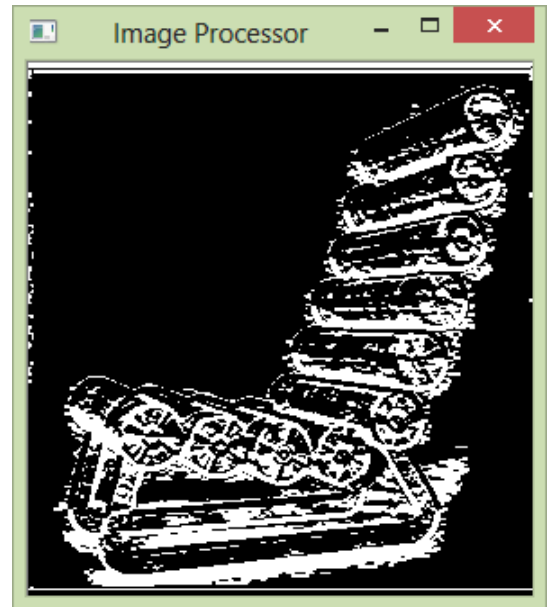
Seperti berikut ini:



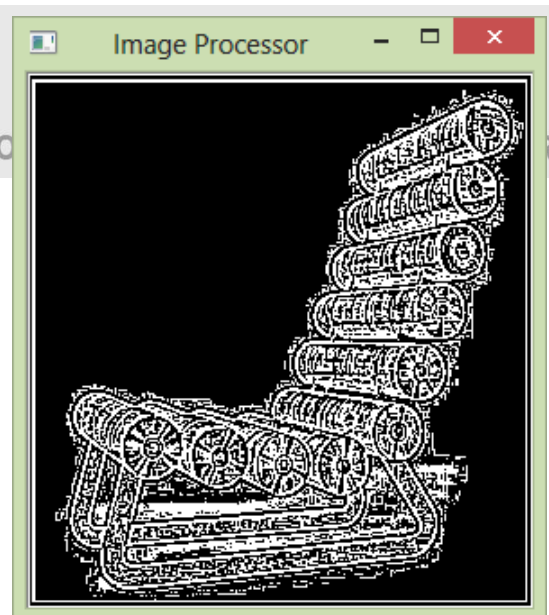
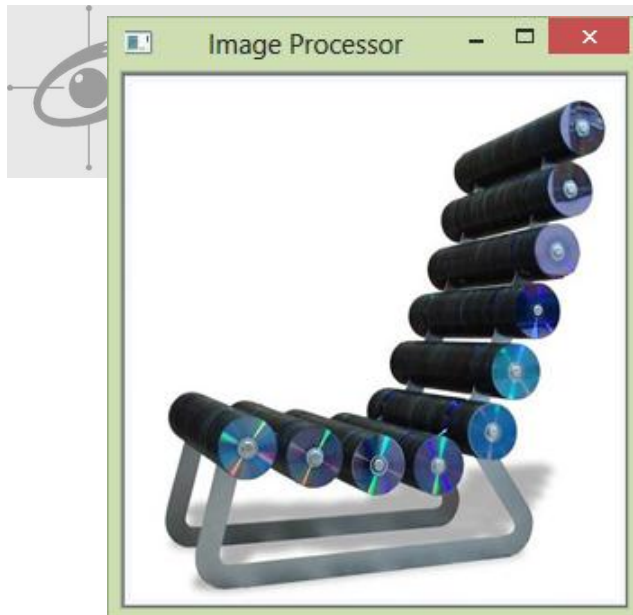
Hasil dari Canny Edge Detection:



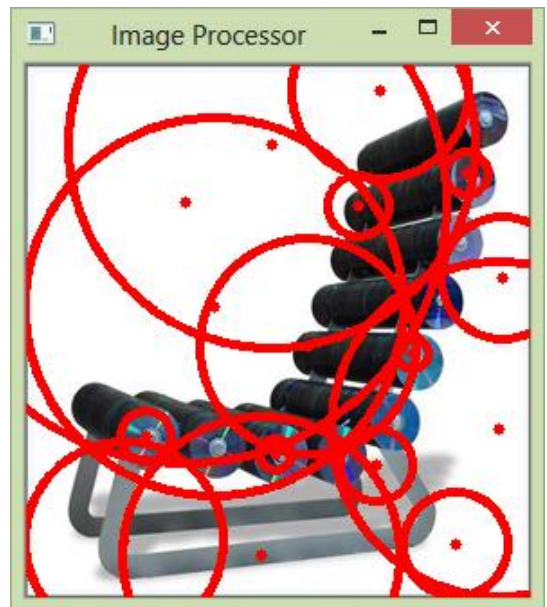
Hasil dari **Sobel Edge Detection** :



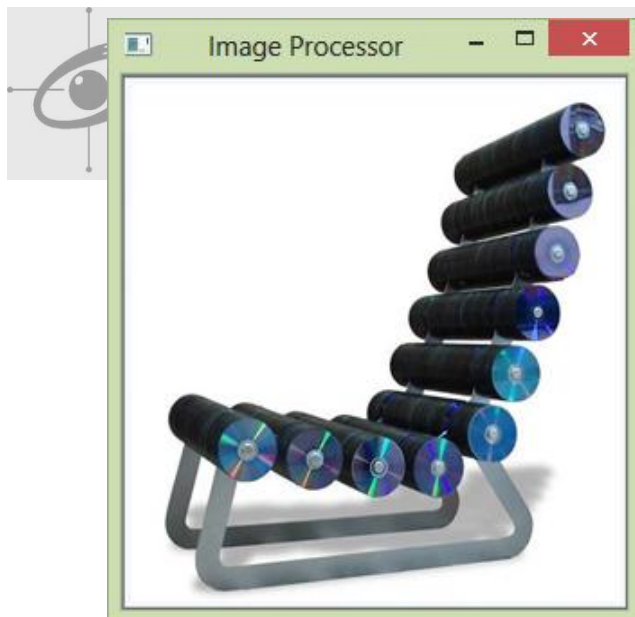
Hasil dari **Laplace Edge Detection** :



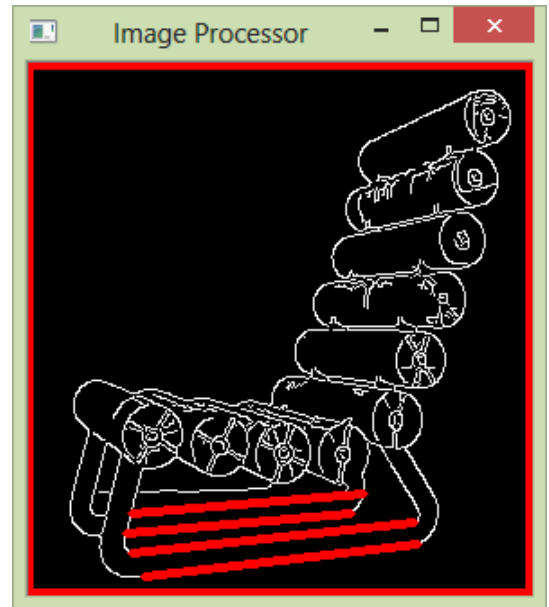
Hasil dari **Circle Detection** :



Hasil dari **Box Detection** :



Hasil **Line Detection** :



Jawaban:

Langkah-langkah pengerjaan:

1. Buat *project* baru dari **Microsoft Visual Studio 2010 C++**. Cara membuat *project* baru masih sama seperti yang telah kita pelajari pada **bab I**.

2. Tuliskan semua *include file* yang dibutuhkan.

```
//Include FLTK
#include<FL\Fl.H>
#include<FL\Fl_Window.H>
#include<FL\Fl_Box.H>
#include<FL\Fl_Button.H>
#include<FL\Fl_File_Chooser.H>

//Include OpenCV
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\core\core.hpp>
#include<opencv2\imgproc\imgproc.hpp>

using namespace cv;
```

3. Pada latihan bab III ini, kita membutuhkan fungsi tambahan. Fungsi ini ditempatkan di dalam *code* (diletakkan setelah *include*). Berikut adalah fungsi tersebut:

```
// helper function:
// finds a cosine of angle between vectors
// from pt0->pt1 and from pt0->pt2
double angle( Point pt1, Point pt2, Point pt0 )
{
    double dx1 = pt1.x - pt0.x;
    double dy1 = pt1.y - pt0.y;
    double dx2 = pt2.x - pt0.x;
    double dy2 = pt2.y - pt0.y;
    return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)
        *(dx2*dx2 + dy2*dy2) + 1e-10);
}
int thresh = 50, N = 11;
// returns sequence of squares detected on the image.
// the sequence is stored in the specified memory storage
void findSquares(const Mat& image,vector<vector<Point>>&squares)
{
    squares.clear();
    Mat pyr, timg, gray0(image.size(), CV_8U), gray;
    // down-scale and upscale the image to filter out the noise
    pyrDown(image, pyr, Size(image.cols/2, image.rows/2));
    pyrUp(pyr, timg, image.size());
    vector<vector<Point>> contours;
    // find squares in every color plane of the image
    for( int c = 0; c < 3; c++ )
    {
        int ch[] = {c, 0};
        mixChannels(&timg, 1, &gray0, 1, ch, 1);
        // try several threshold levels
        for( int l = 0; l < N; l++ )
        {
            // hack: use Canny instead of zero threshold level.
            // Canny helps to catch squares with gradient shading
            if( l == 0 )
            {
                // apply Canny. Take the upper threshold from slider
                // and set the lower to 0 (which forces edges merging)
                Canny(gray0, gray, 0, thresh, 5);
                // dilate canny output to remove potential
                // holes between edge segments
                dilate(gray, gray, Mat(), Point(-1,-1));
            }
        }
    }
}
```





```

else
{
    //apply threshold if l!=0:
    //tgray(x,y) = gray(x,y) < (l+1)*255/N ? 255 : 0
    gray = gray0 >= (l+1)*255/N;
}
// find contours and store them all as a list
findContours(gray, contours, CV_RETR_LIST,
             CV_CHAIN_APPROX_SIMPLE);
vector<Point> approx;
// test each contour
for( size_t i = 0; i < contours.size(); i++ )
{
    //approximate contour with accuracy proportional
    //to the contour perimeter
    approxPolyDP(Mat(contours[i]), approx,
                 arcLength(Mat(contours[i]), true)*0.02, true);
    //square contours should have 4 vertices after
    //approximation relatively large area (to filter
    //out noisy contours) and be convex.
    //Note: absolute value of an area is used because
    //area may be positive or negative - in accordance
    //with the contour orientation
    if( approx.size() == 4 &&
        fabs(contourArea(Mat(approx))) > 1000 &&
        isContourConvex(Mat(approx)) )
    {
        double maxCosine = 0;

        for( int j = 2; j < 5; j++ )
        {
            //find the maximum cosine of the angle
            //between joint edges
            double cosine = fabs(angle(approx[j%4],
                                       approx[j-2], approx[j-1]));
            maxCosine = MAX(maxCosine, cosine);
        }

        // if cosines of all angles are small
        // (all angles are ~90 degree) then
        // write quandrangle vertices to resultant
        // sequence
    }
}

```

```

        if( maxCosine < 0.3 )
            squares.push_back(approx);
    }
}
}
}
}

```

4. Buat semua *object* dan *variable* yang akan digunakan pada *global scope*:


```

//Declare UI component
Fl_Window *window;
Fl_Box *title, *transform_label, *detect_label;
Fl_Button *button[8];
Fl_File_Chooser *chooser;

//Declare OpenCV image object
Mat original, edited;

```

5. Di dalam **int main**, buatlah baris *code* seperti berikut:



```

//Create the widget
window = new Fl_Window(330, 250, "Image Detection");
title = new Fl_Box(10, 0, 300, 30, "Image Detection");
transform_label = new Fl_Box(20, 80, 100, 30, "Edge Detection");
detect_label = new Fl_Box(20, 160, 100, 30, "Shape Detection");
button[0] = new Fl_Button(20, 40, 90, 30, "Load Image");
button[1] = new Fl_Button(120, 40, 90, 30, "Original");
button[2] = new Fl_Button(20, 120, 90, 30, "Canny");
button[3] = new Fl_Button(120, 120, 90, 30, "Sobel");
button[4] = new Fl_Button(220, 120, 90, 30, "Laplace");
button[5] = new Fl_Button(20, 200, 90, 30, "Circle");
button[6] = new Fl_Button(120, 200, 90, 30, "Box");
button[7] = new Fl_Button(220, 200, 90, 30, "Line");
chooser = new Fl_File_Chooser("../", "Image File(*.{jpg,png})",
    Fl_File_Chooser::SINGLE, "Choose Image");

//Deactive button 1-7
for(int i=1;i<8;i++)
{
    button[i]->deactivate();
}

```



```

//Set callback to button
button[0]->callback(load_image);
button[1]->callback(original_image);
button[2]->callback(canny);
button[3]->callback(sobel);
button[4]->callback(laplace);
button[5]->callback(circle);
button[6]->callback(box);
button[7]->callback(line);

title->labelsize(28);
title->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

window->end();
window->show(argc,argv);
return Fl::run();

```

6. Buatlah fungsi **load_image()** untuk di-*callback* oleh **button[0]** dan **isImageFile()** untuk melakukan pengecekan terhadap ekstensi **.jpg** dan **.png**:



```

int isImageFile(const char *fileName)
{
    int len = strlen(fileName);

    if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'j' &&
        tolower(fileName[len-2])== 'p' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }else if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'p' &&
        tolower(fileName[len-2])== 'n' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }
    else return 0;
}

void load_image(Fl_Widget *w, void *v)
{
    chooser->show();
    while(chooser->shown())
    {
        Fl::wait();
    }
}

```

```

    }
    if(chooser->value()!=0 && isImageFile(chooser->value()))
    {
        original = imread(chooser->value());
        imshow("Image Processor", original);

        for(int i=0;i<8;i++)
        {
            button[i]->activate();
        }
    }
}

```

7. Buatlah sebuah fungsi **original_image()** untuk di-*callback* oleh *button* **button[1]**. Digunakan untuk memunculkan gambar **original**:

```

void original_image(Fl_Widget *w, void *v)
{
    imshow("Image Processor", original);
}

```

8. Buatlah sebuah fungsi **canny()** untuk melakukan deteksi tepi menggunakan metode **canny**, yang akan dipanggil oleh **button[2]**:

```

void canny(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Canny(edited, edited, 50, 100, 3);
    imshow("Image Processor", edited);
}

```

Penjelasan:

```

cvtColor(original, edited, CV_RGB2GRAY);

```

Baris *coding* diatas digunakan untuk melakukan *grayscale*.

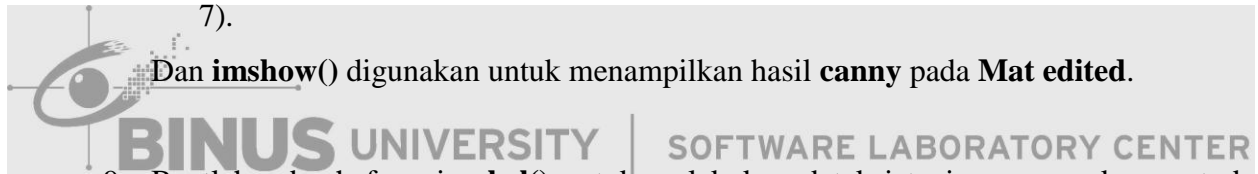
Untuk menggunakan **canny edge detection**, gambar harus diubah menjadi bentuk *grayscale* dengan menggunakan fungsi **cvtColor()**. Metode deteksi **canny** menggunakan turunan pertama dari *filter Gaussian*, yang hasilnya menjadi versi *blur*

sehingga hasilnya tidak dipengaruhi oleh *pixel* gangguan (*noise*) dalam jumlah yang wajar.

```
Canny(edited, edited, 50, 100, 3);
imshow("Image Processor", edited);
```

Fungsi **Canny()** digunakan untuk mendeteksi tepi sebuah gambar menggunakan metode **canny**. Fungsi **Canny()** memiliki parameter:

- 1) **Source**: data gambar berasal.
- 2) **Edge**: data gambar hasil yang menunjukkan hasil berupa deteksi sisi.
- 3) **Lowest thres**: nilai *threshold* terendah. Biasanya memiliki perbandingan 1:2 atau 1:3 dengan nilai *threshold* terbesar.
- 4) **Highest thres**: nilai *threshold* terbesar.
- 5) **Aperture size**: besaran *matrix* untuk parameter besaran matrik **canny** yang digunakan (nilai default adalah 3. Nilai lain yang dapat digunakan adalah 1, 3, 5, 7).



Dan **imshow()** digunakan untuk menampilkan hasil **canny** pada **Mat edited**.

9. Buatlah sebuah fungsi **sobel()** untuk melakukan deteksi tepi menggunakan metode **sobel**, yang akan digunakan untuk dipanggil oleh **button[3]**:

```
void sobel(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Sobel(edited, edited, CV_8U, 1, 2);
    imshow("Image Processor", edited);
}
```

Penjelasan:

```
cvtColor(original, edited, CV_RGB2GRAY);
```

Baris *coding* di atas digunakan untuk melakukan *grayscale*.

```
Sobel(edited, edited, CV_8U, 1, 2);
imshow("Image Processor", edited);
```

Fungsi **Sobel()** menerima parameter berupa:

- 1) **Source**: data gambar berasal.
- 2) **Edge**: data gambar hasil yang menunjukkan hasil berupa deteksi sisi.
- 3) **Depth**: kedalaman warna dari gambar hasil.
- 4) **XOrder**: *Order* derivatif dari X (biasa bernilai 0 untuk keadaan normal dan **YOrder** bernilai 1, ataupun sebaliknya).
- 5) **YOrder**, *Order* derivatif dari Y (Apabila bernilai 1 untuk keadaan normal, maka **XOrder** akan bernilai 0, ataupun sebaliknya).

Dan **imshow()** digunakan untuk menampilkan hasil **sobel** pada **Mat edited**.

10. Buatlah sebuah fungsi **laplace()** untuk melakukan deteksi tepi menggunakan metode **laplace**, yang akan digunakan untuk dipanggil oleh **button[4]**:



```
void laplace(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Laplacian(edited, edited, CV_8U, 3);
    imshow("Image Processor", edited);
}
```

Penjelasan:

```
cvtColor(original, edited, CV_RGB2GRAY);
```

Baris *coding* diatas digunakan untuk melakukan *grayscale*.

```
Laplacian(edited, edited, CV_8U, 3);
imshow("Image Processor", edited);
```


Fungsi **Laplacian()** menerima parameter berupa:

- 1) **Source**: data gambar berasal.
- 2) **Destination**: data gambar tujuan.
- 3) **Depth**: kedalaman warna dari gambar hasil.

- 4) **Aperture size**: besaran matriks untuk parameter besaran matriks **laplacian** yang digunakan (Nilai default adalah 3. Nilai lain yang dapat digunakan adalah 1, 3, 5, 7).

Dan **imshow()** digunakan untuk menampilkan hasil **sobel** pada **Mat edited**.

11. Buatlah sebuah fungsi **circle()** untuk melakukan deteksi terhadap bentuk lingkaran, untuk dipanggil oleh **button[5]**:



```
void circle(Fl_Widget *w, void *v)
{
    vector<Vec3f> circles;
    cvtColor(original, edited, CV_RGB2GRAY);
    GaussianBlur(edited, edited, Size(9,9),2,2);
    HoughCircles(edited,circles,CV_HOUGH_GRADIENT,1,edited.rows/8);
    edited = original.clone();

    for(size_t i = 0;i<circles.size();i++)
    {
        Point center(cvRound(circles[i][0]),cvRound(circles[i][1]));
        int radius = cvRound(circles[i][2]);
        circle(edited, center, 3, Scalar(255,0,0), -1, 8, 0 );
        circle(edited, center, radius, Scalar(255,0,0), 3, 8, 0 );
    }
    imshow("Image Processor", edited);
}
```

Penjelasan:

```
vector<Vec3f> circles;
```

Membuat *object* dari **vector<Vec3f>**.

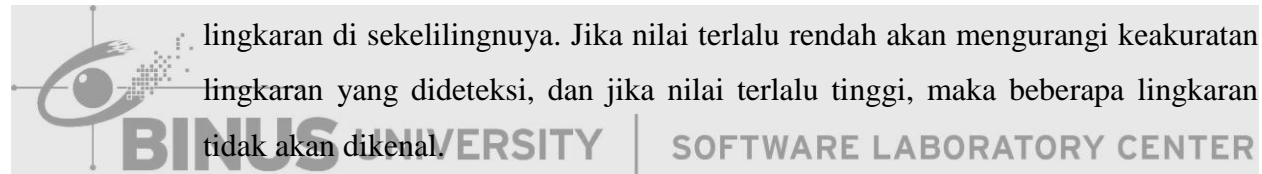
```
cvtColor(original, edited, CV_RGB2GRAY);
GaussianBlur(edited, edited, Size(9,9),2,2);
```

Baris *coding* di atas untuk mendapatkan gambar menjadi *grayscale* dan di-*blur*.

```
HoughCircles(edited,circles,CV_HOUGH_GRADIENT,1,edited.rows/8);
```

Fungsi **HoughCircles()** digunakan untuk mengenali bentuk lingkaran, fungsi ini akan dikenali berupa **vector<Vec3f>**. Dimana setiap isi dari vektor memiliki elemen sendiri yang dapat digambar ulang menjadi lingkaran. Parameter yang diterima oleh **HoughCircles()** adalah:

- 1) **Source**: data gambar asal.
- 2) **Storage**: tempat penyimpanan data.
- 3) **Metode**: biasanya digunakan **CV_HOUGH_GRADIENT** (belum ada metode lain).
- 4) **dp**: resolusi dari akumulator yang digunakan untuk mendeteksi titik tengah lingkaran. Jika $dp = 1$ maka akumulator memiliki resolusi sama dengan gambar masukan, jika $dp = 2$ maka akumulator akan memiliki nilai 2 kali lebih kecil dari ukuran tinggi dan lebarnya.
- 5) **Minimum distance**: diisi nilai jarak minimum dari titik tengah lingkaran dengan lingkaran di sekelilingnya. Jika nilai terlalu rendah akan mengurangi keakuratan lingkaran yang dideteksi, dan jika nilai terlalu tinggi, maka beberapa lingkaran tidak akan dikenal.



```
edited = original.clone();
```

Code diatas digunakan untuk menyiapkan data gambar yang akan dimanipulasi sebagai penanda lingkaran dengan meng-clone gambar asal.

```
for(size_t i = 0;i<circles.size();i++)
{
    Point center(cvRound(circles[i][0]),cvRound(circles[i][1]));
    int radius = cvRound(circles[i][2]);
    circle(edited, center, 3, Scalar(255,0,0), -1, 8, 0 );
    circle(edited, center, radius, Scalar(255,0,0), 3, 8, 0 );
}
imshow("Image Processor", edited);
```

Looping digunakan untuk melakukan perulangan sebanyak jumlah dari lingkaran (jumlah lingkaran didapatkan dari *object circles* yang telah menerima hasil operasi dari fungsi **HoughCircles()**).

circle() digunakan untuk menggambar lingkaran sederhana. Memiliki parameter berupa:

- 1) **Source**: data gambar yang akan digambar.
- 2) **Titik tengah lingkaran**: biasa digunakan fungsi **Point()** untuk mencari titik tengah dari lingkaran yang akan digambar. **Point()** menerima parameter berupa titik x dan titik y, akan berfungsi layaknya untuk mencari titik tengah dari kedua koordinat tersebut. Pada baris *coding* di atas digunakan fungsi **cvRound()** untuk melakukan pembulatan terhadap titik x dan titik y.
- 3) **Radius**: radius dari lingkaran.
- 4) **Warna**: warna dari lingkaran yang akan digambar.
- 5) **Ketebalan**: ketebalan dari lingkaran yang akan digambar.

6) **Tipe garis** yang akan digambar:

- **8 - 8-connected line.**
- **4 - 4-connected line.**
- **CV_AA - antialiased line.**

7) **Shift**: jumlah bit fraksional dipusat koordinat dan nilai jari-jari.

Dan **imshow()** digunakan untuk menampilkan hasil.

12. Buatlah sebuah fungsi **box()** untuk melakukan deteksi terhadap bentuk kotak, untuk dipanggil oleh **button[6]**:

```

void box(Fl_Widget *w, void *v)
{
    vector<vector<Point>> squares;
    findSquares(original, squares);
    original.copyTo(edited);
    for(size_t i=0;i<squares.size();i++)
    {
        const Point* p = &squares[i][0];
        int n = (int)squares[i].size();
        polylines(edited,&p,&n,1,true,Scalar(255,0,0),3,CV_AA);
    }
    imshow("Image Processor", edited);
}

```

Penjelasan:

```
vector<vector<Point>> squares;
```

Membuat *object* dari **vector<Vec3f>**.

```
findSquares(original, squares);
```

Fungsi **findSquares()** memiliki parameter berupa:

- 1) **Source**: data asal gambar yang akan dimanipulasi.
- 2) **Storage**: tempat penyimpanan data.

Catatan: Fungsi **findSquares()** berasal dari fungsi tambahan yang berada setelah *include*.

```
original.copyTo(edited);
```

Code diatas digunakan untuk menyiapkan data gambar yang akan dimanipulasi sebagai penanda lingkaran dengan meng-*copy* dari gambar asal.

```

for(size_t i=0;i<squares.size();i++)
{
    const Point* p = &squares[i][0];
    int n = (int)squares[i].size();
    polylines(edited,&p,&n,1,true,Scalar(255,0,0),3,CV_AA);
}
imshow("Image Processor", edited);

```

Looping digunakan untuk melakukan perulangan sebanyak jumlah dari *box* (jumlah *box* didapatkan dari *object squares* yang telah menerima hasil operasi dari fungsi **findSquares()**).

polylines() digunakan untuk menggambar persegi. Memiliki parameter berupa:

- 1) **Source**: data gambar yang akan digambar.
- 2) **Curves**: persegi yang akan digambarkan.
- 3) **Vertex count**: jumlah titik dari box.
- 4) **Number of curves**: jumlah persegi yang digambarkan tiap 1 lokasi.
- 5) **Close status**: status ditutup atau tidak. Menggunakan tipe data *bool* sehingga hanya menerima *true* atau *false* saja.
- 6) **Warna**: warna dari persegi yang akan digambar.
- 7) **Ketebalan**: ketebalan dari persegi yang akan digambar.
- 8) **Tipe garis** yang akan digambar:
 - **8 - 8-connected line.**
 - **4 - 4-connected line.**
 - **CV_AA - antialiased line.**

Dan **imshow()** digunakan untuk menampilkan hasil.

13. Buatlah sebuah fungsi **line()** untuk melakukan deteksi terhadap bentuk kotak, yang akan dipanggil oleh **button[7]**:


```

void line(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Canny(edited, edited, 50, 200, 3);
    vector<Vec4i> lines;
    HoughLinesP(edited, lines, 1, CV_PI/180, 80, 30, 10);
    original.copyTo(edited);
    for(size_t i=0; i<lines.size();i++)
    {
        line(edited,Point(lines[i][0],lines[i][1]),
              Point(lines[i][2],lines[i][3]),Scalar(255,0,0),3,8);
    }
    imshow("Image Processor", edited);
}

```

Penjelasan:

```

cvtColor(original, edited, CV_RGB2GRAY);
Canny(edited, edited, 50, 200, 3);

```

Baris coding di atas digunakan untuk membuat *image* menjadi *grayscale* lalu dilakukan operasi **canny** (*object eddited*).

```

HoughLinesP(edited, lines, 1, CV_PI/180, 80, 30, 10);

```

Fungsi **HoughCirclesP()** digunakan untuk mengenali bentuk garis, fungsi ini akan dikenali berupa **vector<Vec3f>**. Dimana setiap isi dari vektor memiliki elemen sendiri yang dapat digambar ulang menjadi garis. Parameter yang diterima oleh **HoughCirclesP()** adalah:

- 1) **Source**: data gambar asal.
- 2) **Storage**: tempat penyimpanan data.
- 3) **Rho**: jarak resolusi pixel antara unit.
- 4) **Theta**: resolusi sudut diukur dalam radian.
- 5) **Threshold**: sebuah baris dikembalikan oleh fungsi apa bila nilai akumulator yang sesuai lebih besar dari ambang.
- 6) **Min length**: panjang minimum dari garis.
- 7) **Max gap**: jarak maksimum antar garis.


```
original.copyTo(edited);
```

Code diatas digunakan untuk menyiapkan data gambar yang akan dimanipulasi sebagai penanda garis dengan meng-*copy* dari gambar asal.

```
for(size_t i=0; i<lines.size();i++)
{
    line(edited,Point(lines[i][0],lines[i][1]),
        Point(lines[i][2],lines[i][3]),Scalar(255,0,0),3,8);
}
imshow("Image Processor", edited);
```

Looping digunakan untuk melakukan perulangan sebanyak jumlah dari garis (jumlah garis didapatkan dari *object lines* yang telah menerima hasil operasi dari fungsi **HoughCirclesP()**).

line() digunakan untuk membuat garis pada gambar. Menerima *parameter* berupa:

- 1) **Source:** data gambar yang akan dimanipulasi.
- 2) **Titik pertama.**
- 3) **Titik kedua.**
- 4) **Warna garis.**
- 5) **Ketebalan.**
- 6) **Tipe garis.**

Berikut ini adalah keseluruhan *code* dari *code* di atas:

```
//Include FLTK
#include<FL\Fl.H>
#include<FL\Fl_Window.H>
#include<FL\Fl_Box.H>
#include<FL\Fl_Button.H>
#include<FL\Fl_File_Chooser.H>

//Include OpenCV
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\core\core.hpp>
#include<opencv2\imgproc\imgproc.hpp>

using namespace cv;
```

```

// helper function:
// finds a cosine of angle between vectors
// from pt0->pt1 and from pt0->pt2
double angle( Point pt1, Point pt2, Point pt0 )
{
    double dx1 = pt1.x - pt0.x;
    double dy1 = pt1.y - pt0.y;
    double dx2 = pt2.x - pt0.x;
    double dy2 = pt2.y - pt0.y;
    return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)
        *(dx2*dx2 + dy2*dy2) + 1e-10);
}
int thresh = 50, N = 11;
// returns sequence of squares detected on the image.
// the sequence is stored in the specified memory storage
void findSquares(const Mat& image,vector<vector<Point>>&squares)
{
    squares.clear();
    Mat pyr, timg, gray0(image.size(), CV_8U), gray;
    // down-scale and upscale the image to filter out the noise
    pyrDown(image, pyr, Size(image.cols/2, image.rows/2));
    pyrUp(pyr, timg, image.size());
    vector<vector<Point>> contours;
    // find squares in every color plane of the image
    for( int c = 0; c < 3; c++ )
    {
        int ch[] = {c, 0};
        mixChannels(&timg, 1, &gray0, 1, ch, 1);
        // try several threshold levels
        for( int l = 0; l < N; l++ )
        {
            // hack: use Canny instead of zero threshold level.
            // Canny helps to catch squares with gradient shading
            if( l == 0 )
            {
                // apply Canny. Take the upper threshold from slider
                // and set the lower to 0 (which forces edges merging)
                Canny(gray0, gray, 0, thresh, 5);
                // dilate canny output to remove potential
                // holes between edge segments
                dilate(gray, gray, Mat(), Point(-1,-1));
            }
            else
            {

```

```

        //apply threshold if l!=0:
        //tgray(x,y) = gray(x,y) < (l+1)*255/N ? 255 : 0
        gray = gray0 >= (l+1)*255/N;
    }
    // find contours and store them all as a list
    findContours(gray, contours, CV_RETR_LIST,
        CV_CHAIN_APPROX_SIMPLE);
    vector<Point> approx;
    // test each contour
    for( size_t i = 0; i < contours.size(); i++ )
    {
        //approximate contour with accuracy proportional
        //to the contour perimeter
        approxPolyDP(Mat(contours[i]), approx,
            arcLength(Mat(contours[i]), true)*0.02, true);
        //square contours should have 4 vertices after
        //approximation relatively large area (to filter
        //out noisy contours) and be convex.
        //Note: absolute value of an area is used because
        //area may be positive or negative - in accordance
        //with the contour orientation
        if( approx.size() == 4 &&
            fabs(contourArea(Mat(approx))) > 1000 &&
            isContourConvex(Mat(approx)) )
        {
            double maxCosine = 0;

            for( int j = 2; j < 5; j++ )
            {
                //find the maximum cosine of the angle
                //between joint edges
                double cosine = fabs(angle(approx[j%4],
                    approx[j-2], approx[j-1]));
                maxCosine = MAX(maxCosine, cosine);
            }

            // if cosines of all angles are small
            // (all angles are ~90 degree) then
            // write quadrangle vertices to resultant
            // sequence
            if( maxCosine < 0.3 )
                squares.push_back(approx);
        }
    }

```

```

    }
}

//Declare UI component
Fl_Window *window;
Fl_Box *title, *transform_label, *detect_label;
Fl_Button *button[8];
Fl_File_Chooser *chooser;

//Declare OpenCV image object
Mat original, edited;

int isImageFile(const char *fileName)
{
    int len = strlen(fileName);

    if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'j' &&
        tolower(fileName[len-2])== 'p' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }else if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'p' &&
        tolower(fileName[len-2])== 'n' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }
    else return 0;
}

void load_image(Fl_Widget *w, void *v)
{
    chooser->show();
    while(chooser->shown())
    {
        Fl::wait();
    }
    if(chooser->value()!=0 && isImageFile(chooser->value()))
    {
        original = imread(chooser->value());
        imshow("Image Processor", original);
    }
}

```



```

        for(int i=0;i<8;i++)
        {
            button[i]->activate();
        }
    }
}

void original_image(Fl_Widget *w, void *v)
{
    imshow("Image Processor", original);
}

void canny(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Canny(edited, edited, 50, 100, 3);
    imshow("Image Processor", edited);
}

void sobel(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Sobel(edited, edited, CV_8U, 1, 2);
    imshow("Image Processor", edited);
}

void laplace(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Laplacian(edited, edited, CV_8U, 3);
    imshow("Image Processor", edited);
}

void circle(Fl_Widget *w, void *v)
{
    vector<Vec3f> circles;
    cvtColor(original, edited, CV_RGB2GRAY);
    GaussianBlur(edited, edited, Size(9,9),2,2);
    HoughCircles(edited,circles,CV_HOUGH_GRADIENT,1,edited.rows/8);
    edited = original.clone();

    for(size_t i = 0;i<circles.size();i++)
    {
        Point center(cvRound(circles[i][0]),cvRound(circles[i][1]));
        int radius = cvRound(circles[i][2]);
    }
}

```

```

        circle(edited, center, 3, Scalar(255,0,0), -1, 8, 0 );
        circle(edited, center, radius, Scalar(255,0,0), 3, 8, 0 );
    }
    imshow("Image Processor", edited);
}

void box(Fl_Widget *w, void *v)
{
    vector<vector<Point>> squares;
    findSquares(original, squares);
    original.copyTo(edited);
    for(size_t i=0;i<squares.size();i++)
    {
        const Point* p = &squares[i][0];
        int n = (int)squares[i].size();
        polylines(edited,&p,&n,1,true,Scalar(255,0,0),3,CV_AA);
    }
    imshow("Image Processor", edited);
}

void line(Fl_Widget *w, void *v)
{
    cvtColor(original, edited, CV_RGB2GRAY);
    Canny(edited, edited, 50, 200, 3);
    vector<Vec4i> lines;
    HoughLinesP(edited, lines, 1, CV_PI/180, 80, 30, 10);
    original.copyTo(edited);
    for(size_t i=0; i<lines.size();i++)
    {
        line(edited,Point(lines[i][0],lines[i][1]),
            Point(lines[i][2],lines[i][3]),Scalar(255,0,0),3,8);
    }
    imshow("Image Processor", edited);
}

int main(int argc, char **argv)
{
    //Create the widget
    window = new Fl_Window(330, 250, "Image Detection");
    title = new Fl_Box(10, 0, 300, 30, "Image Detection");
    transform_label = new Fl_Box(20, 80, 100, 30, "Edge Detection");
    detect_label = new Fl_Box(20, 160, 100, 30, "Shape Detection");

```

```

button[0] = new Fl_Button(20, 40, 90, 30, "Load Image");
button[1] = new Fl_Button(120, 40, 90, 30, "Original");
button[2] = new Fl_Button(20, 120, 90, 30, "Canny");
button[3] = new Fl_Button(120, 120, 90, 30, "Sobel");
button[4] = new Fl_Button(220, 120, 90, 30, "Laplace");
button[5] = new Fl_Button(20, 200, 90, 30, "Circle");
button[6] = new Fl_Button(120, 200, 90, 30, "Box");
button[7] = new Fl_Button(220, 200, 90, 30, "Line");
chooser = new Fl_File_Chooser("../", "Image File(*.{jpg,png})",
    Fl_File_Chooser::SINGLE, "Choose Image");

//Deactive button 1-7
for(int i=1;i<8;i++)
{
    button[i]->deactivate();
}

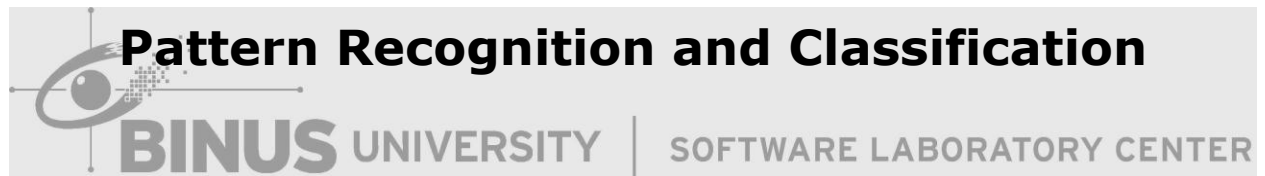
//Set callback to button
button[0]->callback(load_image);
button[1]->callback(original_image);
button[2]->callback(canny);
button[3]->callback(sobel);
button[4]->callback(laplace);
button[5]->callback(circle);
button[6]->callback(box);
button[7]->callback(line);

title->labelsize(28);
title->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

window->end();
window->show(argc,argv);
return Fl::run();
}

```

Chapter 04



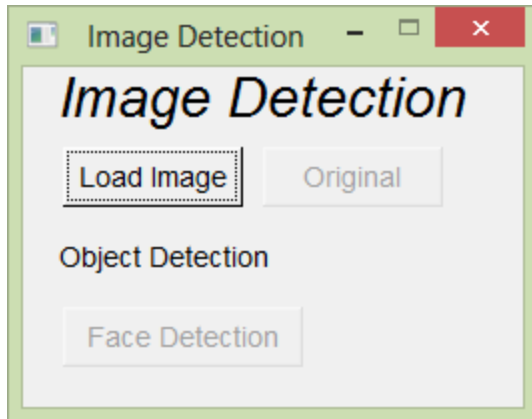
4.1. Pattern Recognition and Classification

Pattern Recognition adalah penugasan semacam nilai *output* (atau label) untuk nilai masukan yang diberikan (atau misalnya), menurut beberapa algoritma tertentu. Sebuah contoh dari **Pattern Recognition** adalah **Classification**, yang mencoba untuk memberikan setiap nilai masukan ke salah satu dari satu *hardware* (misalnya, menentukan apakah *email* yang diberikan adalah "*spam*" atau "*non-spam*"). Namun, **Pattern Recognition** adalah masalah yang lebih umum yang meliputi jenis lain *output* juga. Contoh lain adalah regresi, yang memberikan *output* real-nilai untuk setiap masukan, pelabelan urutan, yang memberikan kelas untuk setiap anggota urutan nilai (misalnya, bagian dari penandaan pidato, yang memberikan bagian dari pidato untuk setiap kata di sebuah kalimat input), dan *parsing*, yang memberikan pohon parse ke input kalimat, menggambarkan struktur sintaksis kalimat.

Di dalam *computer vision* sendiri, pengenalan pola sangat sering digunakan untuk melakukan beberapa deteksi bentuk gambar. Antara lain adalah pengenalan bentuk wajah, bentuk badan, dan lainnya. Hal ini menggunakan metode pengenalan pola (**Pattern Recognition**) yang telah diberikan pengenalan sebelumnya ke dalam suatu jaringan *neural*.

4.2. Exercise

Buatlah sebuah program untuk mendeteksi wajah seperti berikut:



Face Recognition :



Jawaban:

Langkah-langkah pengerjaan:

1. Buat *project* baru dari **Microsoft Visual Studio 2010 C++**. Cara membuat *project* baru masih sama seperti yang telah kita pelajari pada **bab I**.

2. Tuliskan semua *include file* yang dibutuhkan.

```
//Include FLTK
#include<FL\Fl.H>
#include<FL\Fl_Window.H>
#include<FL\Fl_Box.H>
#include<FL\Fl_Button.H>
#include<FL\Fl_File_Chooser.H>

//Include OpenCV
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\core\core.hpp>
#include<opencv2\imgproc\imgproc.hpp>
#include<opencv2\objdetect\objdetect.hpp>

using namespace cv;
```

3. Buat semua *object* dan *variable* yang akan digunakan:



```
//Declare UI component
Fl_Window *window;
Fl_Box *title, *transform_label;
Fl_Button *button[3];
Fl_File_Chooser *chooser;

//Declare OpenCV image object
Mat original, edited;
```

4. Di dalam **int main**, buatlah baris *coding* seperti berikut ini:

```
//Create the widget
window = new Fl_Window(250, 170, "Image Detection");
title = new Fl_Box(20, 0, 200, 30, "Image Detection");
transform_label = new Fl_Box(20, 80, 100, 30, "Object Detection");
button[0] = new Fl_Button(20, 40, 90, 30, "Load Image");
button[1] = new Fl_Button(120, 40, 90, 30, "Original");
button[2] = new Fl_Button(20, 120, 120, 30, "Face Detection");
chooser = new Fl_File_Chooser("../", "Image File(*.{jpg,png})",
    Fl_File_Chooser::SINGLE, "Choose Image");

//Deactive button 1-3
for(int i=1;i<3;i++)
{
```

```

        button[i]->deactivate();
    }

    //Set callback to button
    button[0]->callback(load_image);
    button[1]->callback(original_image);
    button[2]->callback(face_detection);

    title->labelsize(28);
    title->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

    window->end();
    window->show(argc,argv);
    return Fl::run();

```

5. Buatlah fungsi **load_image()** untuk di-*callback* oleh **button[0]** dan **isImageFile()** untuk melakukan pengecekan terhadap ekstensi **.jpg** dan **.png**:



```

int isImageFile(const char *fileName)
{
    int len = strlen(fileName);

    if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'j' &&
        tolower(fileName[len-2])== 'p' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }else if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'p' &&
        tolower(fileName[len-2])== 'n' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }
    else return 0;
}

void load_image(Fl_Widget *w, void *v)
{
    chooser->show();
    while(chooser->shown())
    {
        Fl::wait();
    }
    if(chooser->value()!=0 && isImageFile(chooser->value()))

```

```

{
    original = imread(chooser->value());
    imshow("Image Processor", original);

    for(int i=0;i<3;i++)
    {
        button[i]->activate();
    }
}

```

6. Buatlah fungsi **original_image()** untuk di-*callback* oleh **button[1]**.

```

void original_image(Fl_Widget *w, void *v)
{
    imshow("Image Processor", original);
}

```

7. Buatlah sebuah fungsi **face_detection()** untuk melakukan deteksi wajah, yang akan dipanggil oleh **button[2]**:

```

void face_detection(Fl_Widget *w, void *v)
{
    CascadeClassifier faceClassifier;
    faceClassifier.load("haarcascade_frontalface_alt2.xml");
    vector<Rect> faces;
    original.copyTo(edited);
    faceClassifier.detectMultiScale(edited, faces, 1.1, 2,
                                    0|CV_HAAR_SCALE_IMAGE,
                                    Size(30,30));

    for( int i = 0; i < faces.size(); i++ )
    {
        Point center(faces[i].x + faces[i].width*0.5,
                    faces[i].y + faces[i].height*0.5);
        ellipse(edited, center, Size(faces[i].width*0.5,
                                    faces[i].height*0.5), 0, 0, 360,
                Scalar(255, 0, 255), 4, 8, 0);
    }
    imshow("Image Processor", edited);
}

```

Penjelasan:

Untuk melakukan deteksi wajah menggunakan **OpenCV** dibutuhkan *file* “**haarcascade_frontalface_alt.xml**”. Yang terdapat di dalam *folder* “**OpenCV2.3\opencv\data\haarcascades**”. **XML** yang di-*load* tersebut adalah bagian dari *file* yang digunakan untuk *training* pengenalan wajah yang telah disediakan oleh **OpenCV** sebelumnya. *Training* adalah aktifitas yang dilakukan untuk mendapatkan pengenalan mengenai sesuatu (**Neuro Computing**). Selain untuk mengenali wajah, **OpenCV** menyediakan banyak sekali fasilitas lain untuk mengenali beberapa bagian tubuh, antara lain:

- **haarcascade_frontalface_alt.xml**
- **haarcascade_frontalface_alt_tree.xml**
- **haarcascade_fullbody.xml**
- **haarcascade_profileface.xml**
- **haarcascade_lowerbody.xml**
- **haarcascade_upperbody.xml**

Sedangkan tipe data **CascadeClassifier** adalah *struct* yang digunakan **OpenCV** untuk menyimpan data hasil training tersebut.



Fungsi yang digunakan oleh **OpenCV** sendiri untuk mengenali wajah adalah **CascadeClassifier::detectMultiScale()**. Fungsi ini menerima parameter berupa:

- 1) **Source**: data gambar yang akan dideteksi.
- 2) **Storage**: tempat penyimpanan berupa vektor yang akan digunakan sebagai media penyimpanan proses pengenalan wajah.
- 3) **ScaleFactor**: faktor skala perbesaran dari *window* yang akan digunakan (biasanya diisi 1.1, yang berarti mengalami perbesaran sebesar 10%).
- 4) **minNeighbor**: jarak minimum dari sebuah persegi panjang yang membentuk *object*.
- 5) **Flag**: metode yang digunakan untuk mengenali wajah. Untuk method ini, kita menggunakan **CV_HAAR_SCALE_IMAGE** sebagai metode pengenalan wajah (metode **CV_HAAR_DO_CANNY_PRUNING** dapat digunakan sebagai alternatif lain). Metode ini dipakai untuk memvalidasi gambar yang memiliki tepi terlalu sedikit ataupun terlalu banyak. Apabila telah divalidasi, gambar akan di-

threshold lalu digunakan metode *pruning* digunakan untuk mempercepat proses pencarian wajah.

- 6) **minSize**: ukuran minimum dari wajah yang akan dideteksi. (ukuran *default* yang digunakan dalam *code* ini adalah ukuran sebesar 30x30).

Berikut ini adalah keseluruhan *code* dari *code* di atas:

```
//Include FLTK
#include<FL\Fl.H>
#include<FL\Fl_Window.H>
#include<FL\Fl_Box.H>
#include<FL\Fl_Button.H>
#include<FL\Fl_File_Chooser.H>

//Include OpenCV
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\core\core.hpp>
#include<opencv2\imgproc\imgproc.hpp>
#include<opencv2\objdetect\objdetect.hpp>

using namespace cv;

//Declare UI component
Fl_Window *window;
Fl_Box *title, *transform_label;
Fl_Button *button[3];
Fl_File_Chooser *chooser;

//Declare OpenCV image object
Mat original, edited;

int isImageFile(const char *fileName)
{
    int len = strlen(fileName);

    if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'j' &&
        tolower(fileName[len-2])== 'p' &&
        tolower(fileName[len-1])== 'g'){
        return 1;
    }else if(tolower(fileName[len-4])== '.' &&
        tolower(fileName[len-3])== 'p' &&
        tolower(fileName[len-2])== 'n' &&
        tolower(fileName[len-1])== 'g'){
```

```

        return 1;
    }
    else return 0;
}

void load_image(Fl_Widget *w, void *v)
{
    chooser->show();
    while(chooser->shown())
    {
        Fl::wait();
    }
    if(chooser->value()!=0 && isImageFile(chooser->value()))
    {
        original = imread(chooser->value());
        imshow("Image Processor", original);

        for(int i=0;i<3;i++)
        {
            button[i]->activate();
        }
    }
}

void original_image(Fl_Widget *w, void *v)
{
    imshow("Image Processor", original);
}

void face_detection(Fl_Widget *w, void *v)
{
    CascadeClassifier faceClassifier;
    faceClassifier.load("haarcascade_frontalface_alt2.xml");
    vector<Rect> faces;
    original.copyTo(edited);
    faceClassifier.detectMultiScale(edited, faces, 1.1, 2,
                                    0|CV_HAAR_SCALE_IMAGE,
                                    Size(30,30));
}

```



```

    for( int i = 0; i < faces.size(); i++ )
    {
        Point center(faces[i].x + faces[i].width*0.5,
                     faces[i].y + faces[i].height*0.5);
        ellipse(edited, center, Size(faces[i].width*0.5,
                                     faces[i].height*0.5), 0, 0, 360,
                Scalar(255, 0, 255), 4, 8, 0);
    }
    imshow("Image Processor",edited);
}

int main(int argc, char **argv)
{
    //Create the widget
    window = new Fl_Window(250, 170, "Image Detection");
    title = new Fl_Box(20, 0, 200, 30, "Image Detection");
    transform_label = new Fl_Box(20, 80, 100, 30, "Object Detection");
    button[0] = new Fl_Button(20, 40, 90, 30, "Load Image");
    button[1] = new Fl_Button(120, 40, 90, 30, "Original");
    button[2] = new Fl_Button(20, 120, 120, 30, "Face Detection");
    chooser = new Fl_File_Chooser("../", "Image File(*.{jpg,png})",
                                   Fl_File_Chooser::SINGLE, "Choose Image");

    //Deactive button 1-3
    for(int i=1;i<3;i++)
    {
        button[i]->deactivate();
    }

    //Set callback to button
    button[0]->callback(load_image);
    button[1]->callback(original_image);
    button[2]->callback(face_detection);

    title->labelsize(28);
    title->labelfont(Fl_Labeltype::_FL_SHADOW_LABEL);

    window->end();
    window->show(argc,argv);
    return Fl::run();
}

```