Chapter 6

Introduction to OpenCV

On successful completion of this course, students will be able to:

- Explain the roles of Computer Vision.
- Install OpenCV for image processing.
- Use CANNY Edge detector.

Introduction

Computer vision is the most important technology in the future in the development of intelligent robot. Computer vision is in the simplest terms, computer vision is the discipline of "teaching machines how to see." This field dates back more than forty years, but the recent explosive growth of digital imaging technology makes the problems of automated image interpretation more exciting and relevant than ever. Computer vision and machine vision differ in how images are created and processed. Computer vision is done with everyday real world video and photography. Machine vision is done in oversimplified situations as to significantly increase reliability while decreasing cost of equipment and complexity of algorithms.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems. Examples of applications of computer vision include systems for:

- Navigation, *e.g.*, by an autonomous mobile robot;
- Detecting events, *e.g.*, for visual surveillance or people counting;
- Organizing information, *e.g.*, for indexing databases of images and image sequences;
- Modeling objects or environments, *e.g.*, medical image analysis or topographical modeling;
- Interaction, *e.g.*, as the input to a device for computer-human interaction, and;
- Automatic inspection, *e.g.*, in manufacturing applications.

Computer vision is fast moving towards video data, as it has more information for object detection and localization even though there scale and rotational variance. An essential component of robotics has to do with artificial sensory systems in general and artificial vision in particular. While it is true that robotics systems exist (including many successful industrial robots) that have no sensory equipment (or very limited sensors) they tend to be very brittle systems. They need to have their work area perfectly lit, with no shadows or mess. They must have the parts needed in precisely the right position and orientation, and if they are moved to a new location, they may require hours of recalibration. If a system could be developed that could make sense out of a visual scene it would greatly enhance the potential for robotics applications. It is therefore not surprising that the study of robot vision and intelligent robotics go hand-in-hand.

Introduction of OpenCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. You may download the latest version such as OpenCV 2.4.7.

OpenCV's built in modules are powerful and versatile enough to solve most of your computer vision problems. OpenCV provides you with a set of modules that can execute the functionalities listed in table 6. 1.

Tabel 6.1 *Modules in OpenCV.*

No	Module	Functionality
1	Core	Core data structures, data type and memory management
2	ImgProc	Image filtering, image transformation and shape analysis
3	Highgui	GUI, reading and writing images and video
4	ML	Statistical models and classification algorithms for use in computer vision applications
5	Objdetect	Object detection using cascade and histogram of gradient classifiers
6	Video	Motion analysis and object tracking in video
7	Calib3d	Camera calibration and 3D Reconstruction from multiple views

You need an editor and compiler of Visual Studio. Net 2010/2013 for editing and compiling OpenCV program. You must first configure the Visual C + +. Net where the library files and the source must be included. Some library files must also be added to the linker input in Visual C + +. The steps are:

1) Run the program and extract to, let say f:/OpenCV246.

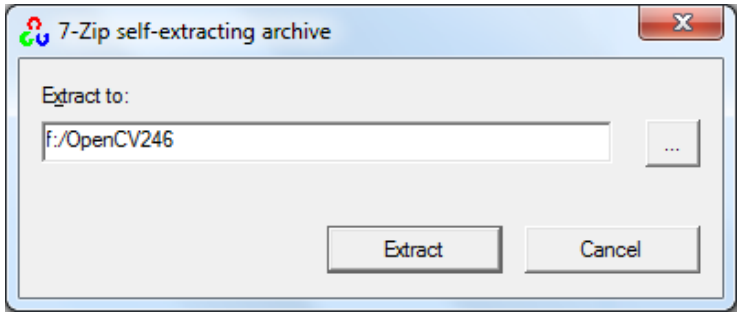

Figure 6.1 *Extracting files to a folder.*

2) Add these paths to your Path Variable:

```
f:\OpenCV246\opencv\build\x86\vc10\bin
f:\OpenCV246\opencv\build\common\tbb\ia32\vc10
```

3) Now we are ready to create a project with OpenCV. In Visual C++ 2010, create a new Win32 console application called IntelligentRobotics. Now right click the project and select Properties. On the left, choose C/C++ and edit the Additional Include Directories. Add these directories:

```
f:\OpenCV246\opencv\build\include\opencv
f:\OpenCV246\opencv\build\include
```

Figure 6.2 *configuring additional include directories.*

4) Now choose Linker and add this directory to the Additional Library Directories. You need to replace x86 with x64 if you want to build a 64 bit application.

```
f:\OpenCV246\opencv\build\x86\vc10\lib
```

5) Now open the Linker group (press the + sign before it) and select Input. Add these lines to the Additional Dependencies:

```
opencv_core246d.lib
opencv_imgproc246d.lib
opencv_highgui246d.lib
opencv_ml246d.lib
opencv_video246d.lib
opencv_features2d246d.lib
opencv_calib3d246d.lib
opencv_objdetect246d.lib
opencv_contrib246d.lib
opencv_legacy246d.lib
opencv_flann246d.lib
```

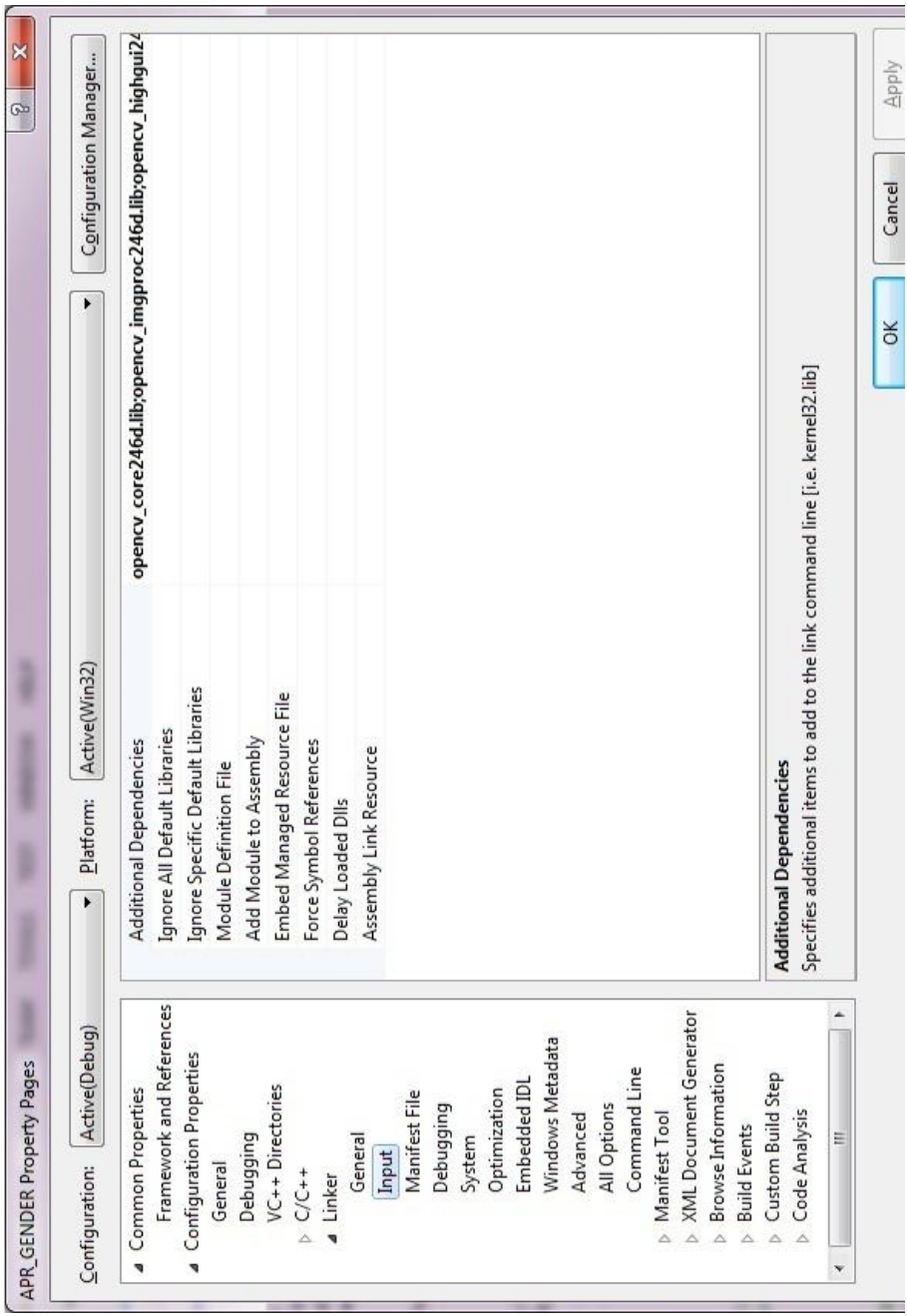

Figure 6.3 *Configuring additional dependencies.*

For example, create a Win32 console application program to display an image in Windows, the following example:

IntelligentRobotics.cpp

```
// Displaying image using cvLoadImage
#include "stdafx.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

int _tmain(int argc, _TCHAR* argv[])
{
 IplImage *img = cvLoadImage("f:\handsome.jpg");
  cvNamedWindow("Intelligent Robotics with OpenCV",1);
  cvShowImage("OpenCV",img);

  cvWaitKey(0);
  cvDestroyWindow("OpenCV ");
  cvReleaseImage(&img);
       return 0;
}
```

Figure 6.4 *Image displayed using OpenCV.*

Or, if you like to use the 2.x C++ style, you can also use:

DisplayImage.cpp

```
// Displaying an image using 2.x C++ style
#include <iostream>
#include <stdio.h>
#include <opencv2/opencv.hpp>
#include <opencv2/highgui//highgui.hpp>

using namespace cv;
using namespace std;
int main( int argc, char** argv )
{
```

```
    // Creating an object img from Mat
    cv::Mat img = cv::imread("f:\handsome.jpg");
    cv::imshow("Modern Robotics with OpenCV",img);
    cv::waitKey(); //wait user hit the keyboard
    return EXIT_SUCCESS;
}
```

Digital Image Processing

An image is an array, or a matrix of square pixels arranged in columns and rows format. A *grayscale image* is composed of pixels represented by multiple bits of information, typically ranging from 2 to 8 bits or more. A *color image* is typically represented by a bit depth ranging from 8 to 24 or higher. With a 24-bit image, the bits are often divided into three groupings: 8 for red, 8 for green, and 8 for blue. Combinations of those bits are used to represent other colors. A 24-bit image offers 16.7 million (2^{24}) color values.

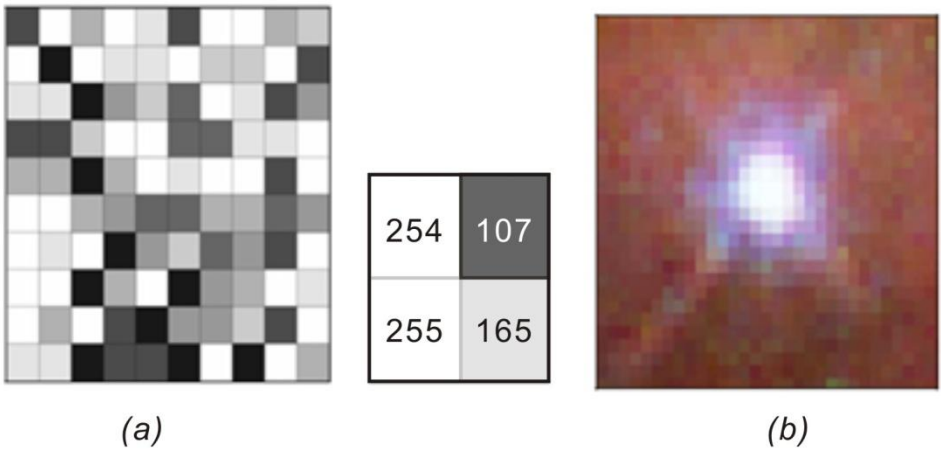

Figure 6.5 *Grayscale image in 8 bit format (a), and truecolor image consist of 3 grayscale image red, green and blue.*

To convert color image to grayscale, since red color has more wavelength of all the three colors, and green is the color that has not only less wavelength then red color but also green is the color that gives more soothing effect to the eyes. It means that we have to decrease the contribution of red color, and increase the contribution of the green color, and put blue color contribution in between these two.

So the new equation in that form is:

```
grayscale image = ( (0.3 * R) + (0.59 * G) + (0.11 * B) ).
```

According to this equation, Red has contributed 33%, Green has contributed 59% which is greater in all three colors and Blue has contributed 11%.

Figure 6.6 *Color image (a) and grayscale image (b).*

The purpose of image processing is divided into 5 groups. They are:

1) Visualization - Observe the objects that are not visible.
2) Image sharpening and restoration - To create a better image.
3) Image retrieval - Seek for the image of interest.
4) Measurement of pattern – Measures various objects in an image.
5) Image Recognition – Distinguish the objects in an image.

As an experiment to know the RGB process, create a new project and name RGB, and create the program below:

RGB.cpp:

```
//Adding an RGB
#include "stdafx.h"
#include <stdio.h>
#include <cv.h>
#include <highgui.h>

void sum_rgb( IplImage* src, IplImage* dst ) {
 // Allocate individual image planes.
IplImage* r = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1 );
IplImage* g = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1 );
```

```
IplImage* b = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1 );

// Temporary storage.
IplImage* s = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1 );

// Split image
cvSplit( src, r, g, b, NULL );

// Add equally weighted rgb values.
cvAddWeighted( r, 1./3., g, 1./3., 0.0, s );
cvAddWeighted( s, 2./3., b, 1./3., 0.0, s );

// Truncate the value above 100.
cvThreshold( s, dst, 150, 100, CV_THRESH_TRUNC );

cvReleaseImage( &r );
cvReleaseImage( &g );
cvReleaseImage( &b );
cvReleaseImage( &s );
}

int main(int argc, char** argv) {
// Buat jendela
cvNamedWindow( argv[1], 1 );
// Load the image from the given file name.
IplImage* src = cvLoadImage( argv[1] );
IplImage* dst = cvCreateImage( cvGetSize(src), src->depth, 1);
sum_rgb( src, dst);

// show the window
cvShowImage( argv[1], dst );
// Idle until the user hits the "Esc" key.
while( 1 ) { if( (cvWaitKey( 10 )&0x7f) == 27 ) break; }

// clean the window
cvDestroyWindow( argv[1] );
cvReleaseImage( &src );
cvReleaseImage( &dst );
}
```

The result is an image that its RGB value has changed as shown below:

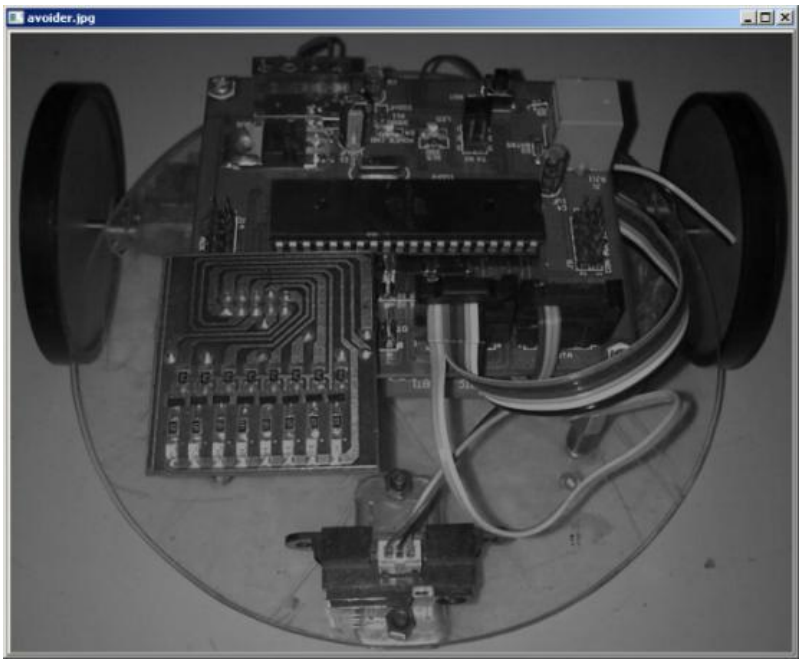

Figure 6.7 *Result of adding RGB value.*

Edge Detection

Edge detection is a technique to locate the edges of objects in the scene. This can be useful for locating the horizon, the corner of an object, white line following, or for determing the shape of an object. The algorithm is quite simple:

- sort through the image matrix pixel by pixel;
- for each pixel, analyze each of the 8 pixels surrounding it;
- record the value of the darkest pixel, and the lightest pixel;
- if (darkest_pixel_value - lightest_pixel_value) > threshold);
- then rewrite that pixel as 1;
- else rewrite that pixel as 0.

The *Canny Edge detector* was developed by John F. Canny in 1986. Also known to many as the *optimal detector*, Canny algorithm aims to satisfy three main criteria:

- Low error rate: Meaning a good detection of only existent edges.
- Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.
- Minimal response: Only one detector response per edge.

CannyEdgeDetector.cpp:

```
//Canny Edge Detector
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;

/// Global variables
Mat src, src_gray;
Mat dst, detected_edges;

int edgeThresh = 1;
int lowThreshold;
int const max_lowThreshold = 100;
int ratio = 3;
int kernel_size = 3;
char* window_name = "Canny Edge Detector";

void CannyThreshold(int, void*)
{
 /// Reduce noise with a kernel 3x3
 blur( src_gray, detected_edges, Size(3,3) );

 /// Canny detector
 Canny( detected_edges, detected_edges, lowThreshold,
lowThreshold*ratio, kernel_size );

 /// Using Canny's output as a mask, we display our result
 dst = Scalar::all(0);

 src.copyTo( dst, detected_edges);
 imshow( window_name, dst );
 }
```

```
int main( int argc, char** argv )
{
 /// Load an image
 src = imread("lena.jpg" );

 if( !src.data )
 { return -1; }

 /// Create a matrix of the same type and size as src (for dst)
 dst.create( src.size(), src.type() );

 /// Convert the image to grayscale
 cvtColor( src, src_gray, CV_BGR2GRAY );

 /// Create a window
 namedWindow( window_name, CV_WINDOW_AUTOSIZE );

 /// Create a Trackbar for user to enter threshold
 createTrackbar( "Min Threshold:", window_name, &lowThreshold,
max_lowThreshold, CannyThreshold );

 /// Show the image
 CannyThreshold(0, 0);

 /// Wait until user exit program by pressing a key
 waitKey(0);

 return 0;
 }
```

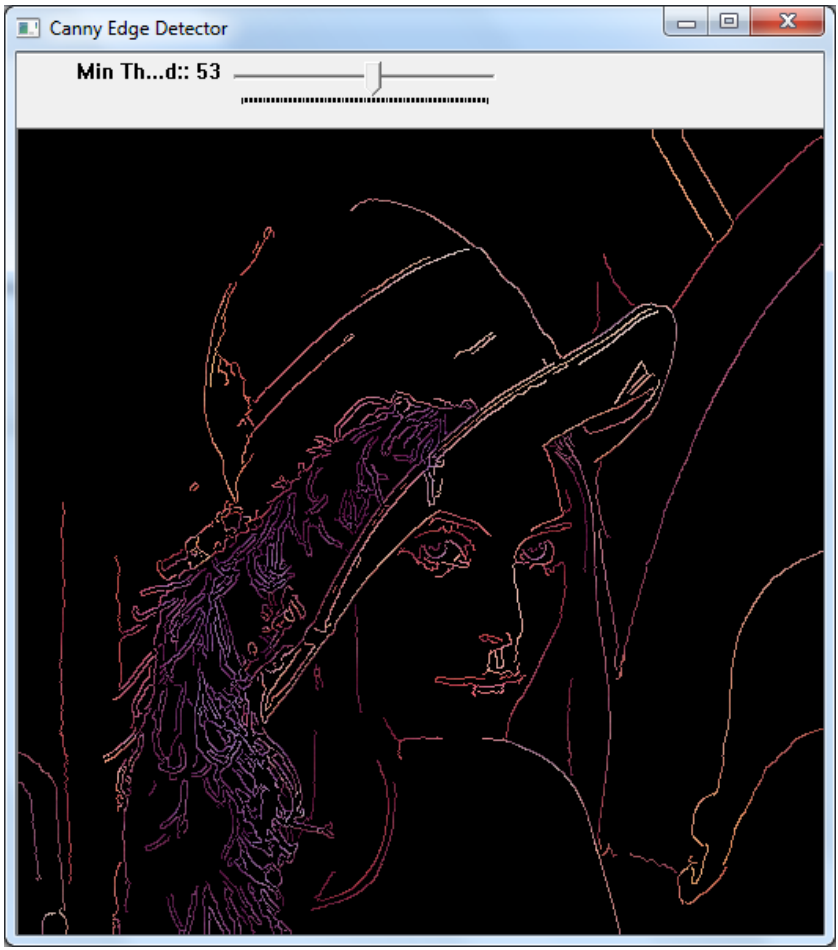

Figure 6.8 *Edge detection using CANNY.*

In image processing, to take the most important areas of an image, commonly known as the ROI (region of interest), can use the following functions:

```
cvSetImageROI(src, cvRect(x,y,width,height));
```

ROI.cpp:

```
#include "stdafx.h"
#include <cv.h>
#include <highgui.h>
```

```
int main(int argc, char** argv) {

 IplImage* src;
 cvNamedWindow("Contoh awal", CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Contoh akhir", CV_WINDOW_AUTOSIZE);
 if( argc == 7 && ((src=cvLoadImage(argv[1],1)) != 0 ))
 {
         int x = atoi(argv[2]);
         int y = atoi(argv[3]);
         int width = atoi(argv[4]);
         int height = atoi(argv[5]);
         int add = atoi(argv[6]);
         cvShowImage( "Contoh awal", src);
         cvSetImageROI(src, cvRect(x,y,width,height));
         cvAddS(src, cvScalar(add),src);
         cvResetImageROI(src);
         cvShowImage( "Contoh akhir",src);
         cvWaitKey();
 }
cvReleaseImage( &src );
cvDestroyWindow("Contoh awal");
cvDestroyWindow("Contoh akhir");
return 0;
}
```

Figure 6.9 *ROI of image.*

Optical Flow

Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene. calcOpticalFlowPyrLK calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.

OpticalFlow.cpp:

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>
#include <cmath>

using namespace cv;
using namespace std;

int main(int argc, char** argv)
{
```

```
  // Load 2 image
  Mat imgA = imread("left02.jpg", CV_LOAD_IMAGE_GRAYSCALE);
  Mat imgB = imread("left03.jpg", CV_LOAD_IMAGE_GRAYSCALE);
  Size img_sz = imgA.size();
  Mat imgC(img_sz,1);

  int win_size = 15;
  int maxCorners = 20;
  double qualityLevel = 0.05;
  double minDistance = 5.0;
  int blockSize = 3;
  double k = 0.04;
  std::vector<cv::Point2f> cornersA;
  cornersA.reserve(maxCorners);
  std::vector<cv::Point2f> cornersB;
  cornersB.reserve(maxCorners);

  goodFeaturesToTrack( imgA,cornersA,maxCorners,qualityLevel,minD
istance,cv::Mat());
  goodFeaturesToTrack( imgB,cornersB,maxCorners,qualityLevel,minD
istance,cv::Mat());

  cornerSubPix( imgA, cornersA, Size( win_size, win_size ),
Size( -1, -1 ),
  TermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, 0.03 ) );
  cornerSubPix( imgB, cornersB, Size( win_size, win_size ),
Size( -1, -1 ),
  TermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, 0.03 ) );

  // Call Lucas Kanade algorithm

  CvSize pyr_sz = Size( img_sz.width+8, img_sz.height/3 );

  std::vector<uchar> features_found;
  features_found.reserve(maxCorners);
  std::vector<float> feature_errors;
  feature_errors.reserve(maxCorners);
  calcOpticalFlowPyrLK( imgA, imgB, cornersA, cornersB,
features_found, feature_errors ,
  Size( win_size, win_size ), 5,
```

```
  cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, 0.3 ),
0 );

  // Make an image of the results

  for( int i=0; i < features_found.size(); i++ ){
  cout<<"Error is "<<feature_errors[i]<<endl;
  //continue;
  cout<<"Got it"<<endl;
  Point p0( ceil( cornersA[i].x ), ceil( cornersA[i].y ) );
  Point p1( ceil( cornersB[i].x ), ceil( cornersB[i].y ) );
  line( imgC, p0, p1, CV_RGB(255,255,255), 2 );
  }

  namedWindow( "ImageA", 0 );
  namedWindow( "ImageB", 0 );
  namedWindow( "LKpyr_OpticalFlow", 0 );

  imshow( "ImageA", imgA );
  imshow( "ImageB", imgB );
  imshow( "LKpyr_OpticalFlow", imgC );

  cvWaitKey(0);
  return 0;
  }
```

(a) (b) (c)

Figure 6.10 *Result of optical flow program from 2 images.*

References

[1] Gary Bradski & Adrian Kaehler, Learning OpenCV (2008), O'Reilly Publisher.

[2] Robert Laganiere, OpenCV 2 Computer Vision Application Programming Cookbook, 2011.

[3] Richard Szeliski, Computer Vision: Algorithms and Applications, 2010.

[4] Budiharto W., Santoso A., Purwanto D., Jazidie A., A Navigation System for Service robot using Stereo Vision, 11th International conference on Control, Automation and Systems, Kyntext-Korea, pp 101-107, 2011.