OpenCV for Computer Vision Applications

Conference Paper · March 2015

CITATIONS
102

READS
30,992

2 authors:

Naveenkumar Mahamkali
SRM University-AP
14 PUBLICATIONS **145** CITATIONS
SEE PROFILE

Vadivel Ayyasamy
SRM University-AP
167 PUBLICATIONS **1,182** CITATIONS
SEE PROFILE

OpenCV for Computer Vision Applications

M. Naveenkumar
Department of Computer Applications
National Institute of Technology Trichy
Tamilnadu
Email:mnaveenmtech@gmail.com

A. Vadivel
Department of Computer Applications
National Institute of Technology Trichy
Tamilnadu
Email: vadi@nitt.edu

***Abstract*—The aim of image processing is to help the computer to understand the content of an image. OpenCV is a library of programming functions mainly used for image processing. It provides de-facto standard API for computer vision applications. We can solve many real time problems using image processing applications. In this paper, sample real time image processing applications of OpenCV are discussed along with steps.**

Keywords— image processing; OpenCV;Edge Detection; Face Detection;

I. INTRODUCTION

Image processing is a form of signal processing in which the input is an image such as a photograph or video frame, the output is an image or set of characteristics related to image. OpenCV is a library of programming functions mainly used for image processing. It is freely available on the open source Berkely Software Distribution license. It was started as a research project by Intel. OpenCV contains various tools to solve computer vision problems. It contains low level image processing functions and high level algorithms for face detection, feature matching and tracking. Some of the main image processing techniques are given below:

Image Filtering:

It is a technique for modifying or enhancing an image. Image filtering is of two types. The one is linear image filtering, in which, the value of an output pixel is a linear combination of the values of the pixels of the input pixel's neighborhood. The second one is the non-linear image filtering, in which, the value of output is not a linear function of its input.

Image Transformation:

Image transformation generates "new" image from two or more sources which highlight particular features or properties of interest, better than the original input images. Basic image transformations apply simple arithmetic operations to the image data. Image subtraction is often used to identify changes that have occurred between images collected on different dates. Main image transformation methods are

- Hough Transform: used to find lines in an image
- Radon Transform: used to reconstruct images from fan-beam and parallel-beam projection data
- Discrete Cosine Transform: used in image and video compression
- Discrete Fourier Transform: used in filtering and frequency analysis
- Wavelet Transform: used to perform discrete wavelet analysis, denoise, and fuse images

Object Tracking:

Object tracking is the process of locating a object (or multiple objects) over a sequence of images. It is one of the most important components in a wide range of applications in computer vision, such as surveillance, human computer interaction, and medical imaging.

Feature Detection:

A feature is defined as an "interesting" part of an image and features are used as a starting point for many computer vision algorithms. Since features are used as the starting point and main primitives for subsequent algorithms, the overall algorithm will often only be as good as its feature detector. Feature detection is a process of finding specific features of a visual stimulus, such as lines, edges or angle. It will be helpful for making local decisions about the local information contents (image structure) in the image.

The modules of openCV for image processing applications are given below

CORE module contains basic data structures and basic functions used by other modules.
IMGPROC module contains image processing related functions such as linear, non-linear image filtering and geometrical image transformations etc.
VIDEO module contains motion estimation and object tracking algorithms.
ML module contains machine-learning interfaces.
HighGUI module contains the basic I/O interfaces and multi-platform windowing capabilities.

II. SAMPLE IMAGE PROCESSING APPLICATIONS-OPENCV

A. Intruder alarm system using motion dection

We can use this type of alarms to detect or prevent criminal activity. It uses the simple motion detection for detecting the intruder. Motion detection is usually a software-based monitoring algorithm. It signals the camera to begin capturing the event when it detects motions. In image-processing, the image is treated as a two dimensional signal. The video captured by the camera is analyzed by the OPENCV program to detect motion.

Steps for finding motion-detection using OpenCV

1. Capture the video from the camera
```
CvCapture* capture =
cvCaptureFromCAM(CV_CAP_ANY);
```
2. Capture the frame1
```
IplImage* frame1 = cvQueryFrame(capture);
```
3. Capture the frame2
```
IplImage* frame2 = cvQueryFrame(capture);
```
4. Blur the images slightly to reduce the noise
```
blur (frame1, frame1_blur, Size(4, 4));
blur (frame2, frame2_blur, Size(4, 4));
```
5. Find the absolute difference
```
absdiff (frame2_blur, frame1_blur,
result_frame);
```

Flow chart for finding motion-detection

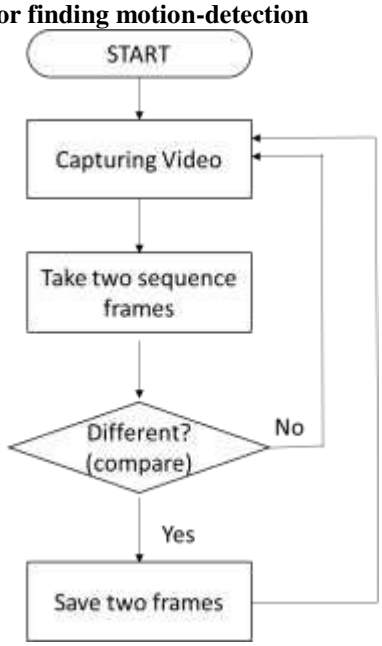

Fig.1. Flow chart for finding motion detection

B. Authentication System using Face Detection

Face authentication is commonly offered as an alternative to passwords for logging into the system. We can write an efficient authentication application using openCV. This application needs web camera to capture the face. The captured image is verified with the images stored in the database of faces. The face recognition involves two phases:

Face Detection, where an input image is searched to find a face, then the image is processed to crop and extracts the person face. OpenCV has face detector called "Haar Cascade classifier". Given an input image, which is from the camera or live video, the face detector examines the image location and classifies it as face or not face. The classifier uses xml file (*haarcascade_frontalface_default.xml)* to decide how to classify each image location. In openCV 2.4.10, the xml file is in path "opencv/sources/data/haarcascades".

Face Recognition is another phase, where the detected face image is compared with images in the database of faces. The openCV framework contains the inbuilt face detector that can work 90-95% on the clear images. It is slightly difficult to detect a face if a person wearing glasses or an image is blurring.

The frames can be grabbed from web camera by using following function.

```
/* Grab the next camera frame. Waits until the
next frame is ready, and provides direct access
to it, so do NOT modify or free the returned
image! It will automatically initialize the
camera on the first frame. */

IplImage* getCameraFrame(CvCapture* &camera)
{
        IplImage *frame;
        int w, h;

        // If the camera hasn't been initialized,
        //then open it.

        if (!camera)
        {
        printf("Acessing the camera ...\n");
        camera = cvCreateCameraCapture( 0 );

        if (!camera)
        {
        printf("Couldn't access the camera.\n");
        exit(1);
        }

        // Try to set the camera resolution to 320
        //x 240.

        cvSetCaptureProperty(camera,
        CV_CAP_PROP_FRAME_WIDTH, 320);
        cvSetCaptureProperty(camera,
        CV_CAP_PROP_FRAME_HEIGHT, 240);
```

```
        // Get the first frame, to make sure the
        //camera is initialized.

        frame = cvQueryFrame( camera );
        if (frame) {
        w = frame->width;
        h = frame->height;
        printf("Got the camera at %dx%d
        resolution.\n", w, h);
        }

        // Wait a little, so that the camera can
        auto-adjust its brightness.
        Sleep(1000); // (in milliseconds)
        }


        // Wait until the next camera frame is
        ready, then grab it.

        frame = cvQueryFrame( camera );
        if (!frame)
        {
        printf("Couldn't grab a camera frame.\n");
        exit(1);
        }
        return frame;
}
```

C. Edge Detection System

It uses the popular edge detection algorithm called canny edge detection. It was developed by John F. Canny in 1986. The structural information can be extracted from the input image by applying this technique. It is a multi-stage algorithm and the stages are presented below.

Noise Reduction:

The edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter.

Finding Intensity Gradient of the Image:

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$Edge - Gradient(G) = \sqrt{G_x^2 + G_y^2} \quad - \quad (1)$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \quad -(2)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

Non-maximum Suppression:

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, it is checked if it is a local maximum in its neighborhood in the direction of gradient and the procedure is depicted in Fig 2.

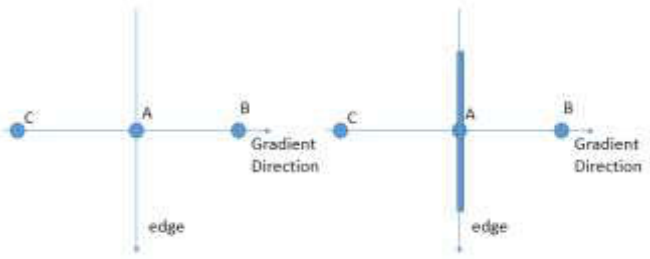

Fig.2. Gradient direction

Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed. In short, the result we get is a binary image with "thin edges".

Hysteresis Thresholding:

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, *minVal* and *maxVal*. Any edges with intensity gradient more than *maxVal* are sure to be edges and those below *minVal* are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded and the procedure is presented in figure 3.

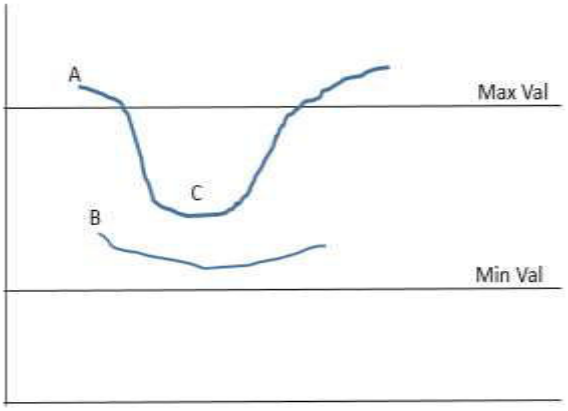

Fig.3. edges between threshold values

The edge A is above the *maxVal*, so considered as "sure-edge". Although edge C is below *maxVal*, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above *minVal* and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select *minVal* and *maxVal* accordingly to get the correct result. This stage also removes small pixels noises on the assumption that edges are long lines. So we finally get strong edges in the image.

openCV put all the above stages in a single function called canny( ). The openCV program for edge detector is as follows.

```
#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 = "Edge Detector";

/**
 * @function CannyThreshold
 * @brief Trackbar callback - Canny
thresholds input with a ratio 1:3
 */
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 );
 }


/** @function main */

int main( int argc, char** argv )
{
  // Load an image
  src = imread( "c:/a.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;
  }
```

The experimental result is presented in Fig.4 & 5.

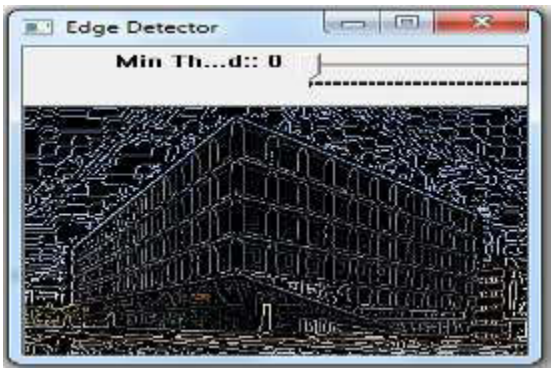

Fig.4. threshold value at zero

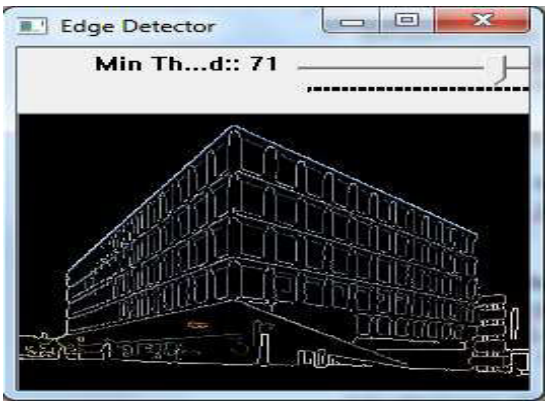

Fig.5. threshold value at seventy one

D. Video Processing using Android Phone

As mobile devices such as smart phones, iPads and tablet pcs are equipped with cameras, the demand of the image processing applications increased. These applications need to be faster and consumes lower power because the mobile device is only powered by a battery. The easy way to increase the performance of the device is to replace the older hardware with newer hardware. The hardware technology depends on the semiconductor technology. If a semiconductor chip contains more number of transistors, the density of transistors is increased. As the density is increased, the leakage of current is also increased. Hence we have to choose an efficient programming language to write an image processing application for the mobile devices.

In these days, android became the famous operating system for the mobile devices. Android developers introduce new application to satisfy the needs of the Smartphone users. Libraries such as OpenGL (Open Graphics Library) and OpenCV (Open Computer Vision) are used for the development of the application [4]. Application that uses camera usually involves an image processing method such as Gaussian, Median, Mean Laplacian, Sobel filter and others. In 2010, a new module is introduced in openCV to deal with GPU acceleration. OpenCV implements a container for images called cv::Mat that exposes access to image raw data. In the GPU module the container cv::gpu::GpuMat stores the image data in the GPU memory.

III. CONCLUSION

The primary interface of OpenCV is written in C++. There are now full interfaces in Python, Java and MATLAB. Wrappers in other languages such as C#, Perl and Ruby have been developed. A CUDA-based GPU interface has been in progress since 2010. OpenCV has in received support from Intel and recently it has received support from willow garage, a privately funded new robotic research institute. openCV can run on different platforms such as windows, android, blackberry, openBSD, ios and linux. Research is going on to introduce new modules in openCV to support robotic perception.

REFERENCES

[1] OpenCV, Open source Computer Vision library. In http://opencv.willowgarage.com/wiki/, 2009

[2] Facial Recognition using OpenCV, Shervin EMAMI1, Valentin Petruț SUCIU2.,www.jmeds.eu.

[3] http://docs.opencv.org/modules/imgproc/doc/imgproc.html

[4] Ammar Anuar, Khairul Muzzammil Saipullah, Nurul Atiqah Ismail, Yewguan Soo "OpenCV Based Real-Time Video Processing Using Android Smartphone" , IJCTEE, Volume 1, Issue 3

[5] http://en.wikipedia.org/wiki/OpenCV

[6] https://blog.cedric.ws/opencv-simple-motion-detection.

[7] http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html