A Preliminary Investigation on Computer Vision for Telemedicine Systems using OpenCV

Zenon Chaczko and
LaReine A. Yeoh
Faculty of Engineering and IT,
University of Technology Sydney
NSW 2007 Australia
zenon.chaczko@uts.edu.au, and
lareine.yeoh.@uts.edu.au

Venkatesh Mahadevan
Faculty of Higher Education,
Swinburne University of Technology
Vic 3140 Australia
vmahadevan@swinburne.edu.au

Abstract—**OpenCV is typically, an open source vision library suitable for computer vision programs. In this paper, we present some of our preliminary investigation experiences of developing Computer Vision programs using OpenCV for robotic telemedicine cluster system, within the practice based ICTD subject within the undergraduate Software Engineering Program at the Faculty of Engineering, University of Technology Sydney (UTS). Firstly, it discusses our shared experiences in designing and implementing Computer Vision subsystem and discusses successes, as well as common problems both experienced and anticipated in adaptation of OpenCV framework and then justifies its purpose building a robotic system for telemedicine. Finally, it attempts to bridge the gap between the theoretical knowledge of design and programming with the practical side of software reuse and modularization when designing and implementing a robotic system for medical applications.**

Keywords- **Computer Vision; OpenCV; TelemedicineCluster,; robotic system; moduliraization.**

I. INTRODUCTION

OpenCV stands for Open Source Computer Vision Library and is designed in C & C++ specifically for increased computational efficiency, supported by most Operating Systems. Example applications of the OpenCV library include Human-Computer Interaction (HCI), Object Identification, Segmentation and Recognition, Face Recognition, Gesture Recognition, Camera and Motion Tracking, Ego Motion, Motion Understanding, Structure From Motion (SFM), Stereo and Multi-Camera Calibration and Depth Computation and Mobile Robotics [4] [5]. On the other hand, the field of Computer Vision is concerned with the transformation of image data from the physical world, captured by an electronic device, into a new representation for processing by an artificial system. Currently it is still an emerging discipline with no standard formulation for how computer vision problems can be solved. Moreover, software systems for image processing can be computationally intensive due to the processing of arrays of data in complex algorithms.

This paper investigates and reports on experiences of teaching Computer Vision for robotic telemedicine cluster system, within the Information and Communication Technology Development (ICTD) subject. Our main goal is to evaluate the outcomes of this experience with the object of following a reflective process at the design and development phases of the subjects to improve the learning and teaching. We will describe methods applied in teaching and learning Computer Vision using OpenCV software libraries. The approach combines best practice aspects, design and algorithmic trade-offs, the Computer Vision problem domain, image processing paradigms, technology and tools as well engineering standards and pedagogy.

The OpenCV library was developed to help ease the burden of developing visioning software by offering a large number of pre-built image processing functions and general-purpose numeric algorithms. The multi-level library design approach taken by OpenCV helps developers tackle the fundamental problem with image processing, where every situation is completely different. Pre-built higher level ready-for-use functions such as cvCalibrateCamera and cvCalcMotionGradient return a reasonable result under most conditions (not all) and are designed to speed up the development process. These high level functions are made from a series of compound or primitive functions all the way down to standard, basic mathematical operators.

The entire range is available to the developer for customization and the creation of unique algorithms to suit the individual's imaging problem. OpenCV libraries can be grouped into different packages in a hierarchy as follows [4]:

- CV & CvAux– compose of the high level components and standard functions, some possessing pre-built algorithms.
- MLL (Machine Learning Library) – consists of 'smart algorithms' based on statistical learning, that focus on turning data into useful information.
- HighGUI –deals solely with displaying images to the user as well as both still-image and video inputs and outputs
- CxCore – sits on the next level down in the hierarchy, it contains the fundamental structures on which OpenCV is built and very basic image algorithms and operations.

978-0-7695-3977-5/10 $26.00 © 2010 IEEE
DOI 10.1109/ICMLC.2010.70

- IPP Integrated Performance Primitives) - allows one to program at a low level to optimize code for Intel CPU architectures.

II. EASE OF USE OF OPENCV FOR COMPUTER VISIONING

One of the challenges to Computer Visioning is the corruption of data by noise, interference and other distortions from sources such as changing weather conditions, lighting, reflections and movement, coming from the analogue world. For a computer to process the data, it needs to be converted into a digital format, which requires sampling the continuous world input. Image processing is an art in itself, though for the purposes of this discussion can be roughly categorized into 3 main areas:

- Image pre-processing
- Image Segmentation (basics leading to Object Detection)
- Colour detection.

Image pre-processing involves performing a number of operations on a raw data set to prepare it for extraction and manipulation. Such image enhancement techniques can be used in either the spatial or frequency domain. This can be done in either the spatial or frequency domain. Frequencies in the fourier transform of an image can be associated with patterns of intensity variations in an image, where low frequencies correspond to a slow variation in pixel intensity, and higher frequencies, rapid level changes in an image [3]. Also filtering in the frequency domain involves the following steps [3]:

1. Input image
2. Do pre-processing
3. Compute the Fourier transform of the image
4. Multiply this by the filter function
5. Compute the inverse Fourier transform
6. Extract the real part of the result
7. Do post-processing
8. Left with the enhanced image

A. Filtering

Gaussian smoothing filtering process can be done in either frequency or spatial. A Gaussian filter or a smoothing filter is a convolution operator that blurs images and removes detail and noise with a Gaussian function, with parameters specified by the developer. The output response of a Gaussian filter is the average of the pixels contained in the neighborhood of the filter mask. Initially one might think that there would be no advantage to blurring an image but in fact this operation is quite useful. The filter acts as a low pass filter and preserves edges while eliminating noise. The Gaussian filter is very useful when detecting edges and corners. **The function `cvSmooth` – also allows for other types of 'smoothening', including MEDIAN, BLUR, BILATERAL. Fig.1 illustrates the effects of a simple smoothing filter [1].

Figure 1. Frame 511; (L) original gray image and ® smoothed image

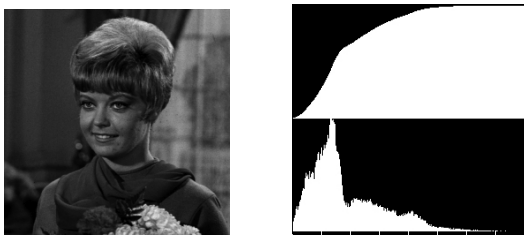
1	1	1
1	1	1
1	1	1

Figure 2. Smoothing filter

A histogram is a plot of the number of pixels in a given image, for any given pixel 'intensity' value or range (called Bins), roughly showing the frequency distribution of the pixels in the image. To transfer the gray levels so that the histogram of the resulting image is equalized to be a constant [6]:

$$h[i] = \text{constant}, \quad \text{for all } i$$

The purpose of the histogram includes equal use of all available gray levels and for further histogram specification. For example, an 8-bit, single channel image, there are 0-255 possible intensity values on the x-axis. If an image's histogram is narrow and tight, (with the majority of pixels falling over a small value range) and if it is skewed (majority of pixels are located at either end of the intensity scale), object detection is more difficult to perform as there is not much spread or contrast across the image [6].



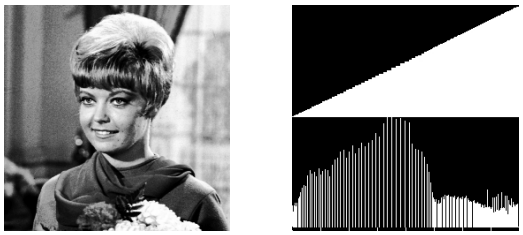


Figure 3. Histrogram equalisation

It is illustrated in Fig.3 that the histogram of a given image is equalized. Although the resulting histogram may not look constant, but the cumulative histogram is an exact linear ramp indicating that the density histogram is indeed equalized. The density histogram is not guaranteed to be a constant because the pixels of the same gray level cannot be separated to satisfy a constant distribution [6].

The histogram equalization technique here can be used to spread out the histogram, effectively evening out the number of pixels with a given value and thus significantly enhancing the contrast in an image, this making edge detection easier. In addition, the first order and second order derivatives in an image have characteristics as follows:

- A first-order derivative generally produces thicker edges in an image [3]
- A second-order derivative has a stronger response to finer details (thin lines and points) – this means that noise is also enhanced onto of finer details.
- First derivatives have a stronger response to a single grey-level step [3]
- Second order derivatives produce a double response to a single grey-level step

B. Edge Detection and Image Segmentation

Image segmentation algorithms in general are based on either of the 2 properties of intensity values [3]:

- Discontinuity – image is partitioned based upon abrupt changes in intensity (eg an edge)
- Similarity – partition image into regions which are similar (set criteria given, eg Thresholding, cvThreshold)

Sobel derivative (cvSobel):

It is used to calculate first, second, third or mixed image derivatives using extended Sobel operator [6]

void cvSobel(const CvArr* src, CvArr* dst, int xorder, int yorder, intaperture_size=3);

src

Source image.

dst

Destination image.

xorder

Order of the derivative x .

yorder

Order of the derivative y.

aperture_size

Size of the extended Sobel kernel, must be 1, 3, 5 or 7. In all cases except 1, aperture_size ×aperture_size separable kernel will be used to calculate the derivative. For aperture_size=1 3x1 or 1x3 kernel is used (Gaussian smoothing is not done). There is also special value CV_SCHARR (=-1) that corresponds to 3x3 [6]

Scharr filter that may give more accurate results than 3x3. Sobel. Scharr aperture is:

$$\begin{vmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{vmatrix}$$

for x-derivative or transposed for y-derivative. The function cvSobel calculates the image derivative by convolving the image with the appropriate kernel [6]:

$$J(x,y) = d^{ox+oy}I/dx^{ox} \bullet dy^{oy} \,|_{(x,y)}$$

The Sobel operators combine Gaussian smoothing and differentiation so the result is more or less robust to the noise. Most often, the function is called with (ox=1, oy=0, apertureSize=3) or (ox=0, oy=1, apertureSize=3) to calculate first x- or y- image derivative [6]. The first case corresponds to

$$\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$$

kernel and the second one corresponds to kernel, depending works by taking the first derivative of the image, therefore the areas where pixels have the greatest intensity change are marked out as edges [6].

$$\begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$$

or

$$\begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}$$

CvSobel calculates derivative by convolving the image with a pre-defined kernel/mask. It also combines Gaussian smoothing and differentiation to reduce the noise level of the final image (no scaling done). It is important to note that the function cvSobel is direction dependent and good for large kernel sizes, better approx without noise, small ones noise limited. kernel, depending on the image origin (origin field of IplImage structure). No scaling is done, so the destination image usually has larger by absolute value numbers than the source image. To avoid overflow, the function requires 16-bit destination image if the source image is 8-bit. The result

can be converted back to 8-bit using cvConvertScale or cvConvertScaleAbs functions. Besides 8-bit images the function can process 32-bit floating-point images. Both source and destination must be single-channel images of equal size or ROI size [6].

Laplace function (cvLaplace) is used to calculate Laplacian of the image

void cvLaplace(const CvArr* src, CvArr* dst, int aperture_size=3);

src

Source image.

dst

Destination image.

aperture_size

Aperture size (it has the same meaning as in cvSobel).

The function cvLaplace calculates Laplacian of the source image by summing second x and y- derivatives calculated using Sobel operator [6]:

dst(x,y) = d2src/dx2 + d2src/dy2

Specifying aperture_size=1 gives the fastest variant that is equal to convolving the image with the following kernel:

| 0 1 0|
| 1 -4 1|
| 0 1 0|

Similar to cvSobel function, no scaling is done and the same combinations of input and output formats are supported. Hence, it is seen that cvLaplace is second order derivative, (ie. zeros count as edges) and it calculates 2nd order Sobel derivative (ie. does Sobel derivative twice), for both x and y directions, and sums both to get the final image (no scaling done) [6].

Edge detection algorithm also called Canny Edge Detector (cvCanny) refined by J Canny in 1986 implements Canny algorithm for edge detection [6]

void cvCanny(const CvArr* image, CvArr* edges, double threshold1,

double threshold2, int aperture_size=3);

image

Input image.

edges

Image to store the edges found by the function.

threshold1

The first threshold.

threshold2

The second threshold.

aperture_size

Aperture parameter for Sobel operator (see cvSobel) [6].

The function cvCanny finds the edges on the input image image and marks them in the output image edges using the Canny algorithm [2] [6]. The smallest of threshold1 and threshold2 is used for edge linking, the largest - to find initial segments of strong edges [2] [6].

Based on the Sobel derivative, as in the first derivatives of the image are computed in both x & y directions and combined into 4 directional derivatives. Where the points meet = local maxima = edges [2] [6]. Therefore it has got similar drawback as the Sobel operator. However, the good thing about Canny is it begins to assemble candidate edge pixels into contours. This is done by applying 'hystersis threshold' under the following conditions [[2] 5] [6]:

If pixel has gradient larger than upper threshold, pixel is part of edge.

- If pixel below lower threshold, rejected completely.
- If pixel in between 2 thresholds, accepted only if it's connected to a pixel above the high threshold.

C. Colour Detection

Colour values can be mapped onto and hence defined by a given colour space. The most common colour spaces used in Computer Visioning are:

- Greyscale – (black and white) for object detection, easier since you only have to work in a single channel
- HSV – (Hue Saturation Value)
  - more intuitive to humans
  - can be defined as two cones put together – light at the top, dark at the bottom
  - Hue is determined by the angle around the cone, and saturation by the radius from the center of the image.
  - Ie. A grayscale image's pixel values will all fall along the axis of rotation.
- RGB – (Red Green Blue)
  - used by most standard electronics
  - Can be defined as a box, with the vertices representing R,G,B + light & dark, in Cartesian style coordinates in 3D space.
  - The closer a value of R,G or B is to zero, the darker it is & the closer it is to 255, the lighter it is.

D. Colour and Histrogram Manipulation

Histogram comparison (cvCompareHist):

The histogram of an image is taken and compared to another existing one.

- Static comparisons of histograms are not always the best and may return false negatives. Instead an adaptive histogram comparison method is preferred, where you take into account the shape of the histogram, so even the image is shifted in colour due to changes in lighting conditions, it can still be detected because the histogram profile remains the same.

- Down side is that a pre-determined set of histograms needs to be stored, if they are not being calculated on the fly.

A similar comparison technique can also be used in 'background subtraction' – where a number of frames (typically in a moving image) are used to differentiate background from the foreground and average out the background.

- Sum over a series of frames to get an average profile of the background histogram.
- Any strange anomalies in the histogram mean that a foreign object is detected or an error condition has occurred.
- Down side is that some form of machine learning is needed/preferable and the ability to store pre-built histograms somewhere in the code/database, however due to the nature of changing conditions the background histogram should always be calculated on the fly as part of a calibration or initialization routine.

III. CONCLUSION

The traditional development of computer systems and robotics do not always include aspects of software modularisation and reuse for building a Computer Vision subsystem. A preliminary application of the mature OpenCV framework may allow the developers not only to build an interesting solution for detection of anomalies in robotised medicine administration but may help them to learn the theory behind the robotic vision and complex image processing algorithms. In this paper we discuss suitability of the OpenCV libraries for providing effective software solutions that could be programmed and tested for applications in telemedicine systems. This preliminary research is a part of ongoing research projects on telemedicine systems at UTS. We plan to extend this project to accommodate further extensive field-based investigations to fully rationalise our findings as part of our future work.

REFERENCES

[1] S. Azary, *Detection of Deformable Objects in a Non-stationary Scene,* Computer Science MS Thesis, Rochester Institute of Technology, NewYork, USA, Also available at http://www.sazary.com/Papers/Thesis_Nov_15_2005.doc [Last Accessed" 19th October, 2009]

[2] G. Bradski, A. Kaehler, 2008, *Learning OpenCV – Computer Vision with the OpenCV Library*, O'Reilly Media Inc. Sebastopol.

[3] R. C. Gonzalez, R. E. Woods, 2002, *Digital Image Processing*, 2 Ed, Prentice Hall Inc., New Jersery.

[4] http://sourceforge.net/project/showfiles.php?group_id=22870 [Last Accessed: 27th May 2009]

[5] OpenCV Wiki, Also available at http://opencv.willowgarage.com/wiki/Welcome, [Last Accessed" 19th October, 2009]

[6] O. O. Hassana, Image Processing, Summer Research Final Report, Stevens Institute of Technology, Department of Defense Telehealth Research Experience, Also available at http://www.stevens.edu/wireless/reu/2005/final-reports/hassana-ozigi-otaru-final-report.pdf, [Last Accessed" 19th October, 2009]