Enhancing Pure Data Interactivity With Computer Vision (Open CV)

Lluis Gómez I Bigórda

Hangar.org

Passatge del Marquès de Santa Isabel, 40

Can Ricart
E-08018 Barcelona

Tel: (+34) 933 084 041

lluisgomez@hangar.org

Yves Degoyon

GISS.tv

c/ Palma de San Just, 7

E-08002 Barcelona

Tel: (+34) 699 502 573

ydegoyon@gmail.com

ABSTRACT

Nowadays Computer Vision is acquiring a growing relevance in the field of interactive arts. The purpose of this paper is to introduce some computer vision techniques which are the base of the actual pd_opencv library, a set of objects (delivered as independent objects and not a library), utilities and examples to use those techniques inside the Pure Data programming language. At the same time we introduce some practical examples of the possible use cases on this topic, and a brief introduction to the internals of the pdp and Gem libraries and the openCV API in order to understand the way to refine and extend the actual pd_opencv approach, as like as a desired routemap for further pd_opencv development.

Keywords

computer vision, gesture recognition, motion tracking, motion detection.

PdCon09, July 19-26, , 2000, São Paulo, SP, Brazil.

1. Introduction

Nowadays Computer Vision is acquiring a growing relevance in the field of interactive arts. From the Myron Krueger's pioneering artwork in the 70's to the present days, lots of artists had been using Computer Vision techniques on their works, extending its field of traditional applications : medical, military, industrial, ... Massively popularized around 2002 with the introduction of the Eye Toy in the game scene, contemporary to the firsts works of Golan Levin, the actual screenario includes lots of interesting pieces in all the interactive and multimedia arts including video dance performances, installations, see for example the popular IR multi-touch screen designs and the acclaimed Reactable musical instrument. In all that time, Computer Vision discipline never stopped to improve.

Pure Data [10] as a programming environment for interactive art has a set of functions and libraries to deal with different interaction paradigms, including HID, networking and communication protocols (osc, midi, dmx, etc), physical computing (i.e. arduino boards) and obviously also computer vision. However, the tools actually included in the graphical libraries for Pure Data (pdp/pidip [7,8] and Gem [9]), are far of the high depth sensing level present in other programming languages. If you take a look at other interactive authoring tools as Open Frameworks, Max/MSP or Proce55ing, you will notice that all of them uses Computer Vision implementations based on the same programming library, namely Open CV [1].

Open CV [1] (Open Source Computer Vision) is a free open-source library of programming functions with more than 500 algorithms mainly aimed at real time computer vision. Open CV is written in performance optimized C/C++ code, it can run on Windows, Linux, and Mac OS X, and is free for commercial and research use under a BSD license. Optionally, when running on an Intel-compatible processor, it makes use of some highly optimized assembly routines : the Intel Integrated Performance Primitives library, IPP [3], designed for an efficient use of the processor features for multi-media processing.

Example applications of the Open CV library are Human-Computer Interaction (HCI), Object Identification, Segmentation and Recognition, Face Recognition, Gesture Recognition, Motion Tracking, Ego Motion, Motion Understanding, Structure From Motion (SFM), Stereo and Multi-Camera Calibration and Depth Computation, Mobile Robotics.

At this point of development, it seems now obvious that a high performance Computer Vision library for Pure Data must be based on Open CV, for its convenient API dedicated to image analysis and for its included mathematical utilities (matrix algebra, ...).

Rather than to rewrite code our proposal will be to provide Open CV bindings for Pure Data, that will give qualitative information that can come out of image and video analysis, in order to include it in a Pure Data processing chain, the data detected by some visual processing will be then able to command vocal synthesis, light control, activation of electronics devices through arduino, ... and, more generally, being a way of controlling all what Pure Data can do.

2. Motivations and Goals

As usual in the PD community, the starting point in the development of new modules reflects an uncovered necessity of the user, who doesn't find a concrete functionality among the existing libraries. If one make a search for existing CV objects in both the most common graphical libraries for pure data (PDP/PiDiP [7,8] and Gem [9]) you will find some useful objects as the pdp_mgrid, pdp_ctrack, pdp_shape and some image morphology objects which are part of pidip, and pix_movement, pix_blob, pix_fiducials in Gem, but these objects are based on very simple image algorithms : the detection of blobs for example common to pix_blob and pdp_shape is a simple analysis of adjacent pixels that forms some shapes or blobs, but this technique is a single-stage analysis technique and can be easily fooled when the contrast in an image is getting blurred and confusion often occurs with such simple techniques.

To access a more sophisticated level of analysis, we must introduce some image representation techniques : Histograms, Fourier Transform, ... and some interpretation techniques inspired by artificial intelligence : Haar's cascade decision tree, image segmentation, contours completion, polygonization, ... which are hopefully all easily available in Open CV (ref : 'Learning OpenCV' [4]).

The goal of the pd_opencv [2] project is to provide a collection of tools, externals and abstractions to perform tasks such as image segmentation, shape and gesture recognition, motion tracking, etc. as well as to provide simple examples to help understanding the basics of computer vision techniques.

pd_opencv [2] wants to be a starting point for a more complex image analysis library, open to research and development inside the pd community.

3. First Considerations Working With Computer Vision

Imagine a "dummy" scenario where a "dummy" artist wants to use the movement of people in front of a camera to, let's say, modify the ambient sound of the room. Motion detection is usually done by frame differencing, this technique attempts to locate moving "objects" in a video sequence by observing the difference on each pixel of the current frame with the corresponding pixel on the previous one. If the pixel color has changed more than a certain threshold value we assume there was some movement in that pixel (coordinates of the image). In the practice, this technique may fail on several situations, for example what happens if we don't have enough light in our room and the camera is producing an image with that typical granular noise? our algorithm will identify that noise as movement.

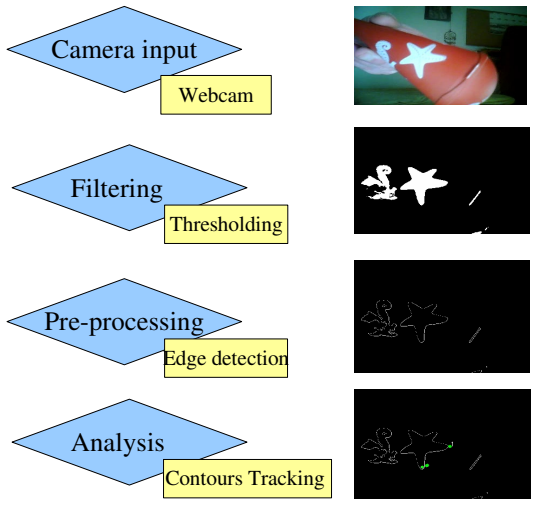
Imagine now the artist also wants to isolate the silhouettes of the visitors to show them projected on the wall. Background subtraction is another basic technique that identifies the pixels of the persons (or objects) in our captured frame according to their difference from a previously recorded background scene taken in the same place without any person (or object) in front of the camera, this is same math that in the movement detection algorithm but comparing each frames with a reference image (and not with the previous frame). This simple pixel by pixel difference operation will work under certain circumstances, but what happens if a person wears a t-shirt with exactly the same color as the background? and what about if the lighting conditions in the room has changed since we recorded our reference background image? these situations will introduce errors in our algorithm results.

As you can see in both the two previous examples, even as them are probably the more elemental CV techniques, those techniques require a well planned capture environment to produce correct results. And that consideration is a common issue for mostly all the CV techniques. In the words of Golan Levin :

<< ... the reliability of computer vision algorithms is limited according to the quality of the incoming video scene, and the definition of a scene's "quality" is determined by the specific algorithms which are used to analyze it, students approaching computer vision for the first time are encouraged to apply as much effort to optimizing their physical scenario as they do to their software code. In many cases, a cleverly designed physical environment can permit the tracking of phenomena that might otherwise require much more sophisticated software.>>

Interactive artists should be "hands on work" with CV to acquire the necessary experience to deal with that issues. Take again our "dummy" scenario for a last example related on that, imagine now our "dummy" artist wants to isolate the visitors silhouettes again but with an extra handicap that is the background image is a wall with a moving image projected on it. Anything we take as a reference background to be compared with captured frames will led our algorithm to a messy result, as the image of the background is not static. Making use of infrared light we'll find a solution for that problem: if we have the scene well illuminated with IR light, using an IR camera we'll be able to capture the scene without the video projection of the wall, as the light of the video projector projecting IR frequencies.

In that sense, a good setup and a fitted video capturing device can be as important as a good and very sophisticated algorithm. For this reason, we modelize an Open CV algorithm in 3 phases : filtering, pre-processing and analysis :



In this example, we want to track the extremities of a sea star form, so we pass through a stage of filtering that extracts only the significant forms of the image, then an edge detection stage that reduces the image to its boundaries and finally we pass the image to a contour tracking object that lets us mark the contours we are interested in, e.g. the extremities of the star.

Here the input device is a simple webcam, but for some setup, especially for live performance environment, you might also sometimes consider using special video devices, like IR cameras or heat-sensitive cameras.

We will now detail the actual state of development of pd_opencv, that implements a few Open CV algorithms, mostly taken from Open CV code samples directory and available as an object for PDP (pdp_opencv_*) or for Gem (pix_opencv_*).

4. Implementation Status (as of 06/2009)

4.1 Filtering modules

As explained in the general introduction, these modules are useful to transform the image in a simpler image to be processed by further analysis modules.

4.1.1 pdp/pix_opencv_threshold

Many of Open CV analysis object (blob detection, contours detection) operate better on binary images, this means on an image that is reduced to the pixels above or below a certain level of intensity.

The object threshold is the most used in an Open CV chain and can binarize images ina variety of modes :

Input Parameters :

mode :

CV_THRESH_BINARY : 255 if src>threshold

CV_THRESH_BINARY_INV : 255 if src<threshold

CV_THRESH_TRUNC : threshold if src>threshold

CV_THRESH_TOZERO : 0 if src<threshold

CV_THRESH_TOZERO_INV : 0 if src>threshold

threshold : value of the threshold for testing pixels.

Output :

4.1.2 pdp/pix_opencv_bgsubstract

This filter is used in motion detection when you want to distinguish moving objects from a static background.

It takes an image as a background reference when receiving a 'SET' message and then compare each incoming frame with that, using a threshold value to compare the pixels.

It is useful if you want to detect objects that are in front of an static background and you want to isolate the foreground silhouettes of moving objects.

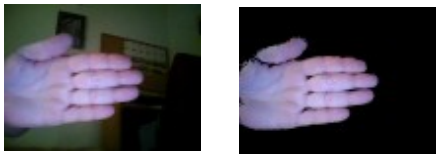
Input Parameters :

SET message :

This is used to capture the background.

threshold : value of the threshold for testing if pixels have changed.

Output :



This object works comparing color values of each pixel, so it can fail if the foreground object has the same color that the background, and also if the light conditions or the shadows changes since you 'SET' the background image.

4.2 Pre-processing modules

This category of objects operates a modification of the input sequence in order to retrieve analysis data in a subsequent stage. They apply some one-stage transformation on the input frames like for example to find adjacent pixels of the same color or something not too complex.

4.2.1 pdp/pix_opencv_morphology

This technique is aimed at distinguishing more precisely the forms in an image using the algorithms of opening/closing and erosion/dilation that increase or reduce the size of zones of bright or dark pixels. It works better on binary image and is helpful to join some zones in a picture that define a wider zone to be processed or tracked.

Input Parameters :

mode : switch between open/close and dilate/erode algorithms.

shape : form of the kernel that is used in the algorithm.

It can be one of : rectangle (0), elliptic (1) or cross-shaped (2).

second inlet : the second inlet indicates the number of iterations that has the effect of accuentating the transformation.

Output :

You see in this example how the zone of texts is 'melted' with the dilation to a single identified zone that can be better processed later.

4.2.2 pdp/pix_opencv_distrans

Another filter that is really close to morphology is the 'distance' algorithm filter that skeletizes the different forms in an image and let's you track a simplified form. It works better on binary images and is particularly appropriate to simplified silhouettes.

Input Parameters :

voronoi (0/1) : activates/desactivates the voronoi triangles partitioning.

type (0/3/5) : optionally uses a mask in the transformation. 0 is none.

second inlet : threshold to detect edges and forms.

Output :

You see here the silhouettes can be isolated using a distrans filter.

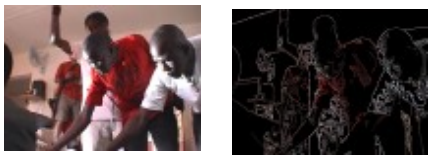
4.2.3 pdp/pix_opencv_edge

This filter is a classical edge detector that detects changes of pixels in the image (using first degree gradient), it can be useful to pre-process an image before passing it to a contour processing analysis object.

Input Parameters :

second inlet : threshold to detect edges.

Output :



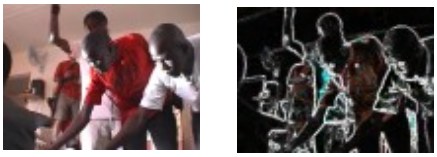
4.2.4 pdp/pix_opencv_laplace

This filter is also an edge detector but it uses a calculation of the second derivative known as Laplace in both directions and detects more accurately the important edges in a picture.

Input Parameters :

second inlet : aperture size (number of points considered to calculate the derivative).

Output :



You can see a Laplace processing outputs a better approach of contours.

4.2.5 pdp/pix_opencv_floodfill

This filter mark some pixel zones that identifies some blobs, e.g zones of adjacent pixels of almost the same intensity and color, taking into account a tolerance in the variation of pixels. It also repaints these zones in a specific color.

Input Parameters :

second inlet : lower tolerance for pixels.

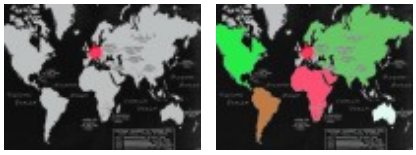
third inlet : upper tolerance for pixels.

connectivity : use 4-points or 8-points connectivity mode.

color : activates/desactivates color mode.

mark messages : mark a certain zone in the frame containing this pixel.

Output :



This object also outputs the coordinates of each detected blobs in the form : 'number x y width height' through its second outlet, so it could be considered as an analysis object, but the blob detection remains a very unstable technique and the tracking is very imprecise with this object.

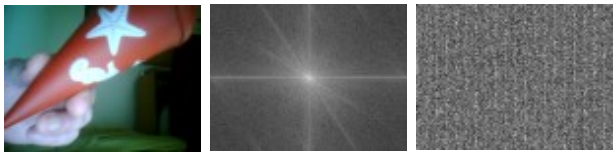
4.2.6 pdp/pix_opencv_dft

This filter is very useful in image analysis as it transform an image from its spacial representation to a frequency domain representation, modelizing an image as an infinite combination of sinusoidal waves. This is know as the Discrete Fourier Transform, very well known of people working in sound processing and it can be useful also in image analysis. It first binarize the incoming images and then calculates the Fourier magnitude and phase diagrams. As it is quite greedy of ressources, it only process a frame when it receives a 'bang' message.

Input Parameters :

bang message : triggers the calculation of a DFT.

Output :



You see here the diagram of the magnitude and the phase of the DFT. The only useful diagram is the magnitude one, that can be compared with the DFT of another frame using binary operators like XOR to detect some patterns in an incoming frame, as shown in the help patch.

4.3 Analysis modules

4.3.1 pdp/pix_opencv_contours_boundingrect

This object calculates the up-right bounding rectangle of all contours of an image. This object works only on binary images distinguishing white pixels zones on a black background.

Input Parameters :

mode : contour detection mode (see cvFindContours [1]).

method : contour detection method (see cvFindContours [1]).

second inlet : minimal size of a contour.

third inlet : maximal size of a contour.

maxmove : maximum movement of a contour between 2 frames (used for identification/numbering of contours).

Output :

This object outputs on its second outlet the position of each contour in the form : “number x y width height” and the total number of contours on its third outlet.

It works detecting any white areas in the input image, it's important to set max/min values of the areas you want to detect to filter non interesting noise or areas, also sometimes you will need to use morphology to transform the contours of the shapes you want to be analyzed.

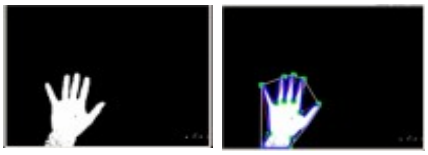
4.3.2 pdp/pix_opencv_contours_convexity

This object looks for the convexity curves of the biggest contour of a binary image. This object considers a contour any group of white pixels on a black background, and therefore, only works on binary images.

Input Parameters :

none.

Output :



This object outputs on its second outlet the number of convexity curves of the biggest contour and, for each convexity curve, it outputs on its third outlet the position of defining points : “number xstart ystart xdepth ydepth xend yend”. This information is kind of dense and can realistically only be used for simple geometrical forms.

This object works detecting the biggest white area in the input image, sometimes you will need to use morphology to transform the contours of the shape to be analyzed.

4.3.3 pdp/pix_opencv_lk

This object detects the most significant points in a contour, enabling to mark peculiar points like angles and asperities. You can then choose the points you want to track by clicking on them.

Input Parameters :

message init : re-calculates contours and most significative points.

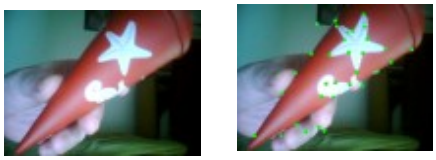
mindistance : minimal distance between points.

quality : quality factor used in approximation (see 'Programming with Open CV [4]).

message mark : mark a point to be tracked.

second outlet : window size for approximation.

Output :



This object outputs on its second outlet the coordinates of the points that you have marked clicking on them, just to be able to output the points that are of interest. It outputs coordinates in the form “number x y”.

4.3.4 pdp/pix_opencv_motempl

This object tracks movement of detected objects using the history of motion on a variable number of frames.

Input Parameters :

second inlet : value of treshold to distinguish objects.

third inlet : minimal size of a detected object.

fourth inlet : maximal size of a detected object.

frame buffer num : number of frames in the motion history (default 4).

min/max delta time : min/max of time to detect motion.

duration : duration of displaying the detected motions.

Output :

This object outputs on its second outlet for each motion component detected, the position, the size of the component and the direction of its movement in the form “number x y width height angle”.

4.3.5 pdp/pix_opencv_histo

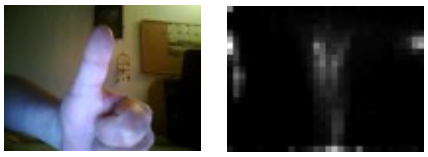
This object memorize and draw the statistical composition of an image in the form of an histogram of the hue and saturation of the pixels. Therefore, it works only on color frames.

You can record up to 80 histograms that defines the composition of the image and by comparing them to the actual frame entering, you can recognize specific configuration of the input image if it has been memorized before. This can work quite well to recognize special gestures like from a human hand and can be used in a kind of gesture command to a pd patch.

Input Parameters :

second outlet : send there a number to memorize a specific histogram.

Output :



This object outputs on its second outlet the number of the closest recorded histogram to the actual incoming frame.

4.3.6 pdp/pix_opencv_haarscascade

This object loads a Haar's cascade decision tree in the form of an XML file, that is based on Haar's technique of dividing an image in squares and calculate some pixels sums on these squares, and the differences between them. Using this technique, we can detect some peculiar form in an image, but this needs to be trained to have the right coefficients stored in the XML with a great number of sample images (> 10.000). Open CV provides a tool to create such a decision tree, but the task is fastidious and not easy. For now, Open CV comes with two examples of these files : one for human faces and one for mouths.

Input Parameters :

load message : load an XML decision tree.

mode (0/1) : use edge detection or not.

minimal size : minimal size of a detected object.

scale factor : scale applied for block size.

Output :

You have to load an .xml decision tree for object recognition.
This object outputs on its second outlet the number of detected objects and on its third outlet, for each detected object, the coordinates and the size of the detected objects in the form “number x y radius”.

5. Programing hints

We give here a few hints to enter into the programmation of Open CV objects for PD in order to open the projects to other developers that are interested in collaborating to this effort.

5.1 Programing for PDP

Open CV works in RGB mode, so every pdp object for Open CV should start converting any incoming frame to an RGB packet with :

```
pdp_packet_convert_ro_or_drop(&x->x_packet0, (int)f, pdp_gensym("bitmap/rgb/*") );
```

Then, some processing in Open CV requires sometimes a Gray image, so you can convert the PDP packet to an Open CV bitmap in two steps : first, copying the PDP packet to an Open CV RGB image :

```
x->image = cvCreateImage(cvSize(x->x_width,x->x_height), IPL_DEPTH_8U, 3);
short int *data   = (short int *)pdp_packet_data(x->x_packet0);
memcpy( x->image->imageData, data, x->x_size*3 );
```

If you need an Open CV Gray image, you can convert it using :

```
x->gray=cvCreateImage(cvSize(x->x_width,x->x_height), IPL_DEPTH_8U, 1);
cvCvtColor(x->image, x->gray, CV_BGR2GRAY):
```

Now you can make all Open CV operations on the RGB or the Gray CV image, when you are done, you can copy the result in a new pdp packet to be propagated, if you don't PDP would modify the incoming image and it's surely not what you want :

```
short int *newdata = (short int *)pdp_packet_data(x->x_packet1);
memcpy( newdata, x->image->imageData, x->x_size*3 );
```

5.1 Programing for GEM

Here it's a bit different as Gem works in RGBA mode, but only if you specify it with a message to a pix_film or pix_video object for example, e.g. to the object that sends the video frame to your Open CV object.

Then, as in PDP you should copy the incoming GEM image in an RGBA Open CV image and eventually convert it to a Gray Open CV image with :

```
orig = cvCreateImage(cvSize(image.xsize,image.ysize), IPL_DEPTH_8U, 4);
gray = cvCreateImage(cvSize(orig->width,orig->height), IPL_DEPTH_8U, 1);
cvCvtColor(orig, gray, CV_RGBA2GRAY);
```

You can then proceed with your Open CV processing and when you're done, you should copy the result to the GEM image again :

```
memcpy( image.data, orig->imageData, image.xsize*image.ysize*4 );
```

6. Conclusion and Perpectives

The Open CV library seems at this date the most complete open source library for computer vision, and it's an assembly of many different techniques and algorithms. We just started a frame for Open CV support in PD, but still haven't explored all its possibilities and litterature, that is huge and we have only read some part of it.

Surprisingly, it's not always the most complex algorithms that work and sometimes a simple technique like the one of the histograms is capable of bringing a sense of intelligence to a pd patch, although it's only a statistical accounting but that, used in a specific context, can be more accurate than some other techniques involving artificial intelligence for example.

Every technique is valid in a certain context, for a specific goal, and it's good to have a variety of techniques to threat the same problem : providing an intelligent way to interact with a patch.

7. References

[1] Open Cv, *an Open Source and optimized library for Computer Vision using various algorithms and artificial intelligence techniques :*

http://sourceforge.net/projects/opencvlibrary/

http://opencv.willowgarage.com/wiki/

[2] PD Open CV development wiki, *a set of objects for Pd/PDP and Pd/Gem, bindings to the Open CV library for Pure Data :*

http://www.hangar.org/wikis/lab/doku.php?id=start:puredata_opencv

[3] Intel's Integrated Performance Primitives (ipp), *optimized and multi-threated primitives for multi-media processing on Intel compatible processors:*

http://software.intel.com/en-us/intel-ipp/

[4] "Learning Open CV" by Gary Rost Bradski, Adrian Kaehler, *a programming guide for Open CV :*

http://my.safaribooksonline.com/9780596516130?portal=oreilly

[5] HIPR2 : Image Processing Learning Ressource, by Robert Fisher, Simon Perkins, Ashley Walker, Erik Wolfart, *a didactic guide to image analysis techniques :*

http://homepages.inf.ed.ac.uk/rbf/HIPR2/

[6] HOWTO write an External for Pure Data, *by Johannes M. Zmölnig, an in-depth programming guide for Pure Data externals:*
http://pdstatic.iem.at/externals-HOWTO/

[7] PureDataPacket (PDP), *Multimedia extension library for PD:*

http://zwizwa.be/pd/pdp/overview.html

[8] PiDiP, *video processing objects for Pure Data Packet*
http://ydegoyon.free.fr/PiDiP.html

[9] GEM, *3D and video extension for Pure Data:*

http://gem.iem.at/

[10] Pure Data, by Miller S. Puckette*, an open-source, BSD licensed multi-media toolbox and programming language:*

http://puredata.info/

[11] Computer Vision for Artists and Designers: Pedagogic Tools and Techniques for Novice Programmers, by Golan Levin and Zachary Lieberman*:*

http://www.flong.com/texts/essays/essay_cvad/