Sehbau: Descriptor Extraction

The use of the program for descriptor extraction is explained, called **dscx**. It extracts the features and describes them geometrically by a few parameters. It generates a number of data files containing: the bounding boxes for the regions; the keypoints of contour segments; the descriptor vectors; and the histograms of attributes (for fast classification).

Read	https://www.researchgate.net/publication/360033329
Repository	https://github.com/Sehbau/DescExtraction
NEXT	matching vectors, https://github.com/Sehbau/MtchVec

The directories in the repository serve for immediate testing with commands as shown in the upcoming sections:

- /**Desc**	example of what the output should look like
- /**Imgs**	sample images for immediate testing
- /**UtilMb**	Matlab scripts to read the output data files

Three types of binaries are available, all 64 bit:
- **dscx.exe**, Windows 10 [x64]
- **dscx_ubu**, Ubuntu 20.04.3 LTS (Focal Fossa) [x64]
- **dscx_deb**, Debian GNU/Linux 11 (bullseye) [ARM]

If the linux binaries fail on your release, then try using 'wine' (windows emulator) and running the windows executable. That would be the easiest for me at the moment. Or let me know with what options I should compile the code (info@sehbau.com).

The use of the program is explained first, and the options that are available for it (Section 1). Then we explain the type of data files that are generated and their format (Section 2).

1 Program Use

Two arguments are required, the image path and the output path for the data files:

```
> dscx pathImg pathOutFile
```

The input image can be jpg or png (or anything offered by https://github.com/nothings/stb). The output path must include a slash, because the program checks for that. Here is an example,

```
> dscx Imgs/img1.jpg  Desc/img1
```

in which the output file name **img1** is chosen to be the same as the image name, for convenience. This will then write the following files into directory **Desc**:

- **img1.Bbox**	bounding boxes of regions (connected components)
- **img1.CntEpt**	endpoints of ridge, river and edge segments
- **img1.vec**	descriptor vectors (attributes)
- **img1.hst**	descriptor histograms

The directory /**UtilMb** contains example scripts for Matlab on how to read the data files. It also contains two example scripts for contour pixel detection and region segmentation, e_CntMap.m and e_Fleckmentation.m, resp. to illustrate the principal techniques for feature detection.

There are no particular image preprocessing algorithms carried out in this program as I have never observed any consistent advantage of using for instance smoothing or low-pass filtering. If you think that your image collection would profit from preprocessing, then carry it out separately (beforehand) and then feed the image to **dscx**.

This is a huge framework. It might fail with unusual images. If you find one, inform me on info@sehbau.com. The program output should terminate with EndOfProgram. It might also fail if you choose unreasonable parameter values, as I have not included reasonability checks everywhere.

1.1 Options

The following (long) options are available, to be specified with a double dash '--'. The first set allows to adjust architecture parameters, the second set contour parameters, the third set region parameters, the fourth set partition parameters and eventually some utility parameters.

Many of the parameters will mainly regulate the number of vectors (saved to **.vec**). For histograms, typically a fuller set of descriptors is taken, as the exact choice of those parameters has lesser influence on computational manipulations. For manipulation with vectors however, some of these parameters can make a huge difference. In particular for place recognition I had observed substantial variations, but a full systematic search is still to be carried out.

The parameter names always start with a lowercase letter and then uses uppercase letters as in the Java notation.

1.1.1 Architecture:

--depth: depth of the segmentation process. Default depth=3. For depth=1 no tree is grown: this corresponds to global thresholding only (with a single threshold). depth=4 can be useful for large images, e.g. larger than 1000 pixel for one image side.

--nLev: number of levels of the pyramid (with downsampling factor equal 2). Default is automatically calculated with top level not smaller than 16 pixels for one map side. For example for a 256x256 image, a five-level pyramid is generated: 256 (original resolution), 128, 64, 32 and 16.

1.1.2 Contours:

--cntMinCtr: contrast threshold for contours. Default =0.05. This is a relative threshold to be set $\in [0.0..1.0]$. It is relative to the largest difference found in the range map for the gray-scale intensity image.

The following two parameters - starting with **skl** - modify the output of the contour selection, the skeleton. This concerns only the vectors - the histograms are generated with all contours without subselection. To understand those changes see plot **ImgPyrSkel.png** [run with **plot**] or used the example script exampleLoadVect.m.

--sklMinSpc: minimum spacing. Default =`0.05`. This is as proportion of the image side length ∈ [0.0..1.0]. Changes here can have a huge effect on performance, recognition accuracy in particular.

--sklMinLen: minimum length. Default =`0.05`. This as well is as proportion of the image side length ∈ [0.0..1.0]. Changes here are less significant, in particular for large spacing values (set with **sklMinSpc**), as then only few short segments remain.

1.1.3 Regions:

--regMinPixNode: minimum number of pixels for a region to be segregated by the thresholding mechanism. This will affect the region count from the 2nd segmentation map on. It will not affect the 1st segmentation map, as that is the input to the 2nd for which segregation starts.

Default equal 6. With larger values, processing occurs more rapidly, but may not segment texture properly anymore.

--rsgMinPix: minimum number of boundary pixels for a radial region descriptor. The number is set for the original image resolution. For higher levels of the pyramid, a correspondingly lower number is used, namely `rsgMinPix-level`. E.g. for a value of 10, the higher pyramidal levels utilize values `9, 8, 7,...`.

Default equal 4 for all levels. For larger values, you risk loosing texture, thus if you interested in the global structure only, it can be useful to set higher values.

--rsgMinCtr: mininum boundary contrast for a radial region descriptor. The value is relative to the average contrast value for all extracted boundaries. Boundaries below that contrast value will not be described as radial descriptor.

1.1.4 Partitions (Arcs/Straighters):

Parameter names starting with **cvp** regulate the partitiong process and therefore affect the outcome of both arc and straighter partitions. Parameter names starting with **arc** and **str** are specific to the respective descriptors.

--cvpMinSiz: minimum boundary size entering the boundary partitioning process. This is relative to the larger image side. For example a value of 0.02 for a [1024 x 2048] image will set the minimum size to 41 pixels. Default equal 0.05.

For small values this will seemingly regulate the number of straigther segments only, as those are rarer. For higher values, it will also start omitting arc segments.

--cvpMinCtr: minimum boundary contrast entering the boundary partitioning process. This is a proportion of the largest boundary contrast determined.

--arcMinLen: minimum arc length to be described. This is relative to image side length. This is a better way of directly regulating the number of arcs than **cvpMinSiz**. Default equal 0.08.

--strMinGer: minimum straightness value for a straighter segment to be accepted. This is relative to the segment itself, namely chord length divided by segment arc length. Default equal 0.8.

1.1.5 Utility:

--prms: displays the parameters used.

--plot: plots contours and region boundaries for the entire pyramid:

-**Icnt.png**: contours plotted onto the color image for the original resolution. This serves to observe the matching between contour type (ridge, river, edge) and the image.

-**ImgPyrBonOnly.png**: boundaries of the entire pyramid and for different depths.
-**ImgPyrCntOnly.png**: contours of the entire pyramid and for different levels.
-**ImgPyrSkel.png**: the selected contours (skeleton).

--verbose: for illustration or for tracking errors.

2 Output Data Files

The following three scripts load the data files using various function scripts:
- **/UtilMb/**`exampleLoadBbox.m`, function scripts in same dir
- **/UtilMb/**`exampleLoadHist.m`, function scripts in **/Hist**
- **/UtilMb/**`exampleLoadVect.m`, function scripts in **/Vect**

The script `exampleLoadBbox.m` also loads the contour endpoints. The following sections give some more explanations.

*2.1 Bounding Boxes (**.Bbox**)*

The bounding boxes for regions are saved in text format to a file with extension **.Bbox**. The list contains all bounding boxes. This allows you to subselect according to your needs.

The bounding boxes are organized as follows. The first two integers hold the number of levels `nLev` and segmentation depth `depth` used for the run. The following numbers hold the region count for each segmentation map, `nBbox`, saved looping levels as the outer loop and looping depth as the inner loop. The example below shows that for two levels and depth equal 3 (using zero-indexing).

```
nLev
depth
nBbox_Lev0_Depth0
nBbox_Lev0_Depth1
nBbox_Lev0_Depth2
nBbox_Lev1_Depth0
nBbox_Lev1_Depth1
nBbox_Lev1_Depth2
```

Then the bounding boxes follow. They are organized analogously to the above inner/outer loop: first all bounding boxes of [lev=0,depth=0], then those of [lev=0,depth=1], etc. A bounding box contains 6 parameters.

```
top, bottom, left, right, area, border       (for lev=0, depth=0)
top, bottom, left, right, area, border
...
top, bottom, left, right, area, border       (for lev=1, depth=2)
```

The parameters describe:

-`top, bottom, left, right`: absolute coordinates that correspond to the map size of the pyramidal level. Thus you need to upsample them by multiplying with the corresponding factor (2, 4, 8, ...).

-`area`: size of bounding box, calculated with the first four parameters.

-`border`: number of touches with the four image sides. The values are:

```
0         no touches: off border
1-4       at one border, directions NESW (top, rite, bot, left)
11-14     at two borders, directions NE,ES,SW,NW (topright, ...)
15,16     "    "     "    , NS axis, WE axis
101-3     at three borders
200       touching all borders
```

The bounding box sizes are typically slightly too small in comparison to annotations in datasets, partly due to the segmentation procedure and partly due to downsampling. Adding margins achieves better annotation correspondence, ie. margin values that correspond to the pyramid level.

Of course you can perform better selections with more information such as boundary contrast and perhaps region attributes. To be included in the future.

2.2 Contour Endpoints (**.CntEpt**)

The points for the contour segments consist of the two endpoints as well as their midpoint. The points are written per level, per contour type and per point type. They are saved in binary format with extension **.CntEpt**.

The first value holds the number of levels. Then each level of the pyramid is written separately with firstly the points of the ridge contours, then those of the river contours and eventually those of the edge contours. The points are written blockwise (and not rowwise as in case of the bounding boxes). The first value holds the number of descriptors. Then follow first all coordinates of the first endpoint (for that level); then all coordinates of the second endpoint; followed by all coordinates for the midpoint. The coordinates `coords` are saved as row/column pairs, per point.

```
nLev
nRdg  (# of ridge contours for lev=0)
[ridge coords of 1st endpoint for lev=0]
[ridge coords of 2nd endpoint for lev=0]
[ridge coords of midpoint     for lev=0]
nRiv  (# of river contours for lev=0)
[river coords of 1st endpoint for lev=0]
[river coords of 2nd endpoint for lev=0]
[river coords of midpoint     for lev=0]
nEdg  (# of edge contours for lev=0)
[edge  coords of 1st endpoint for lev=0]
[edge  coords of 2nd endpoint for lev=0]
[edge  coords of midpoint     for lev=0]
nRdg  (# of ridge contours for lev=1)
[ridge coords of 1st endpoint for lev=1]
[ridge coords of 2nd endpoint for lev=1]
[ridge coords of midpoint     for lev=1]
...
```

As with bounding boxes, the segment coordinates are absolute values corresponding to the map size of the pyramidal level. They need to be upsampled to match the original image resolution if they are used as object/part proposals, and given some spatial width by adding some corresponding value.

2.3 Histograms (**.hst**)

The histograms are saved in binary format with extension **.hst**. They are organized according to increasing dimensionality and spatial order:

- flat, univariate: one-dimensional for the entire image
- flat, bivariate: two-dimensional for the entire image
- spatial, univariate: taken from a 3x3 grid, one-dimensional (univariate)
- spatial, bivariate: taken from a 3x3 grid, two-dimensional (bivariate)

For each type, the order of the descriptor types is: contour, radial signature, arc segment and straighter segment. The reading routine for each of those is in directory **/UtilMb/Hist**.

The radial signature was saved with a number of parameters not described in the (latest) paper (`cncv` to `bis4`). They only slightly improve classification results however.

We recommend starting classification with the first two types, the flat univariate and bivariate histograms. The third type improves accuracy in particular for whole images. The last (fourth) type of histogram, the spatial bivariate, did not consistently improve classification accuracy and can be omitted as it also increases the total dimensionality quite a bit.

2.4 Vectors (**.vec**)

The vector file with extension **.vec** contains the descriptor attributes for all four types (contour, radial signature, arc and straighter segment). It will be loaded by the program matching vectors, **mvec**, to be explained in `https://github.com/Sehbau/MtchVec`.

Most attribute values are normalized to unit range, some only to approximate unit range in case of complex attribute definitions. Angle values come in radians. One should certainly consider scaling, depending on the exact type of manipulation.

This vectorial output could also be tried with Transformer networks.

The example script `exampleLoadVect.m` shows how to read the contours. The parameter values are organized per attribute (dimension), not per (descriptor) vector. Thus, for the purpose of clustering or classification in Matlab (or Python), one has to concatenate them horizontally to a `[nDsc x nAtt]` matrix.

Examples for the other descriptor types will follow later.