Sehbau: Overview Software

An overview of the software suite for creating a visual system is given. It explains the architecture for feature extraction and the presently available descriptors. It introduces the individual programs, carrying out different recognition processes.

Repository `https://github.com/Sehbau`

The system consists of a number of programs that carry out different processes. We firstly explain some basic parameters of the architecture and the descriptors developed so far (Section 1). Then we introduce the core processes, namely descriptor extraction and their matching (Section 2). A learning procedure will be available at some point (Section 3), as well as utility programs for matching and segmentation (Section 4). Most of this is already implemented in *C*, but it will take some time to make it available as individual programs with optional arguments.

This is a huge and novel framework tested on tens of thousand of images, but it might still fail with unusual images. If you happen to encounter one, inform me on `info@sehbau.com`. It should take sizes up to [3000 x 4000], such as a cell phone photo, but have not systematically evaluated such sizes, nor have I tried larger images yet. The program output (**`stdout`**) should terminate with the expression `EndOfProgram`. The program might also fail if you choose unreasonable parameter values, as I have not included reasonability checks everywhere.

I provide utility scripts in Matlab, as that code is easier to read and maintain than Python (less parentheses and dots). For variable notation see also my Computer Vision overview:
`https://www.researchgate.net/publication/336460083`

1 Architecture and Descriptors

For an image, a pyramid of `nLev` levels is generated, ie. `nLev=5` for a 256 x 256 pixels. Each level is processed by a hierarchical segmentation, whose tree `depth` is typically set to 3 for images up to ca. 400 x 500 pixels; for larger images `depth=4` can be useful. This returns an abundance of features, more than in any previous engineering approach, yet still faster due to the use of simplest techniques. These features will be outputted by the program **`dscx`**, one of the core processes.

The features are then partitioned and abstracted resulting in four basic low-dimensional descriptor vectors:

- Contour (**`cnt`**): there are three types of contours: ridge (**`rdg`**), river (**`riv`**) and edge (**`edg`**) contours, all obtained by a simple analysis of the intensity topology.

- Radial shape (**`rsg`**): parameterization based on the radial signature of a region (connected component).

- Arc segment (**`arc`**): curved boundary segments obtained from partitioning the region boundaries.

- Straighter segment (**`str`**): straighter boundary segments obtained from partitioning the region boundaries.

Those basic descriptors are also outputted by program **`dscx`**. Both features and descriptors are suitable for combining with other methodologies (Deep Networks, local features). The descriptors will be matched in programs as introduced next.

The descriptors can be used to perform grouping operations. Many groups have been tested already in Matlab. Their translation to *C* is in progress.

2 Core Processes

The core processes carry out feature extraction and descriptor generation, their matching for recognition and motion estimation.

- **`dscx`** carries out feature extraction and description and outputs some features in raw format (pixels), as well as the descriptor attributes (dimensions).
  `https://github.com/Sehbau/DescExtraction`

- **`mvec`** [beta] matches descriptor vectors and outputs dissimilarity and similarity measurements for each pyramidal layer and descriptor. Can be applied to any structure: shape, object or scene. Beta version. More optional arguments will be included.
  `https://github.com/Sehbau/MtchVec`

- **`motv`** [planned] computes the motion vectors between descriptors of two frames. This is essentially the same as **`mvec`**, but will output in particular the motion vectors between nearest neighbors. That will allow to estimate motion flow.

- **`knnv`** [planned] nearest-neighbor search of structures using a coarse-to-fine strategy to accelerate the retrieval process: it starts with a matching of the top (pyramid) layers, then subselects images and progresses toward finer levels. Early experiments have shown that this strategy not only speeds up the search, but also improves the sorting.

A matching between attribute histograms is not foreseen so far, as for that there exist already statistical packages, such as `sklearn` in Python.

3 Learning

A learning process will be provided that determines category-characteristic descriptors by individual matching of vectors.

- **`kkcan`** [planned]: searches for nearest neighbors across images of one category to find candidate descriptors, based on **`mvec`** essentially.

- **`kkgrp`** [planned]: refines the search and selects final set of category-characteristic descriptors.

- **`kkmtc`** [planned]: matches the category-characteristic descriptors against a new image and outputs the degree of dis-/similarity per category-characteristic descriptor.

4 Utility Processes

The following utility processes will facilitate matching and improve segmentation.

- **brws** [planned]: selects descriptors from a region (focus), specified by the user, and places them into a file for matching with multiple category representations using **knnv**. This is suitable for general scene analysis.

- **srch** [planned]: allows search of an object by specifying one category that then will be matched against different locations in an image.

- **sgrRgb** [planned]: segregates an RGB patch for a given target color, resulting in a black-white image with region boundaries that are more precise than with the gray-scale information as used in program **dscx**.