Sehbau: The Software Suite

C. Rasche
https://github.com/Sehbau

May 12, 2025

This document describes the Sehbau software suite, a computer vision system that operates with contour and region information. The system distinguishes itself from previous approaches by a much faster and richer feature extraction process, that essentially returns all contours and region boundaries. The features are thoroughly parameterized and the resulting parametric description can be used for identification and categorization of any image size and any structure; and for determining motion flow. Those recognition processes can be carried out based on the *same* feature extraction output, and not on multiple feature-extraction phases as in other methodologies, thus enabling to build fast recognition pipelines. The description can certainly be combined with other methodologies such as Deep Networks and Local Features. The software suite is the ideal substrate for building an active vision system. The most complete description of the system is available under:

https://www.researchgate.net/publication/391240551

The software suite comprises a set of program binaries, that carry out different recognition phases such as descriptor extraction, descriptor matching, unsupervised segmentation, retrieval and learning. Examples of how to read the program output, are written in Matlab, because it offers a wider range of plotting tools. A python wrapper is being developed as well. The code notation is explained in Appendix A.

The introduction explains the architecture for feature extraction and the presently available descriptors (Chapter 1). And it surveys the individual programs, both implemented and planned. The starting point for recognition is the program for *descriptor extraction*, called `dscx`, whose file output is explained in Chapter 2. The remaining chapters explain how to match and deploy the descriptor output and how to further analyze a scene. The chapter on applications proposes how to deploy the suite for specific tasks (Chapter 9).

The software is available under,

https://github.com/Sehbau/Haupt

and is presently available for the following compilations, all 64 bit (x86):

Windows	`SEHBAU_win.zip`
Ubuntu, 22.04.4 LTS	`SEHBAU_ubu.tar.gz`, compiled under WSL2
Debian	`SEHBAU_deb.tar.gz`, compiled under WSL2
Fedora	`SEHBAU_fed.tar.gz`, compiled under WSL2
...more in the making...	

This document is also available on:

https://www.researchgate.net/publication/391238505

and that version will contain the latest updates.

Contents

Chapter 1

Introduction

We firstly overview the content of the software suite (Section 1.1), followed by introducing the principal sequence of programs (dscx, mvec, focsel, shpx, etc.), with which one can perform image, focus (region) and shape matching (Section 1.2). Then we explain some basic parameters of the architecture and the descriptors developed so far (Section 1.3). We mention the complete list of programs in three sections: the main processes, namely descriptor extraction and their matching (Section 1.4); the learning programs (Section 1.5); and the utility programs that facilitate matching and segmentation (Section 1.6). Section 1.7 explains the available demonstration programs; Section 1.8 suggest how to deploy the binaries and their output in recognition pipelines.

1.1 Folder Content

The folder /SEHBAU contains the following directories, with blue denoting program binaries, that exist in those directories:

/DemoBaum	demonstration of global-to-local segmentation
/DemoPlcRec	demo for place recognition
/DemoSgrRGB	demo for foreground-background segregation of a RGB image, sgrRGB
/DescExtr	descriptor extraction for an image, dscx, collhimg
/FocSel	focus of attention: selects descriptors from a region, focsel
/MtchHst	matching attribute histograms, mhst
/MtchVec	matching descriptor vectors, mvec, motvec
/ShpExtr	shape extraction for a patch (image), shpx
/ShpMtch	shape matching, mshp
/UtilMb	utility routines for Matlab
/UtilPy	utility routines for Python. In development.
globalsSB.m	global variables (for Matlab)
exsbAll.m	script running all example scripts

As indicated, the program binaries exist in the respective folders. For example the binary for descriptor extraction, called dscx, lies in folder /DescExtr; there is no central folder '/bin'. Example scripts typically contain the prefix exsb; the script exsbAll.m runs all examples. The examples for place recognition contain the prefix

plc. The script globalsSB.m sets mostly paths, but also a few variables, that are necessary for loading the output.

1.2 Survey Processes and Usage

The principal programs are surveyed first (Section 1.2.1), followed by a short introduction of their usage (Section 1.2.2).

1.2.1 Principal Processes and Pipeline

To lauch recognition, we firstly carry out feature extraction and description for the entire image, executed with program dscx. Then we use that image description to match it with stored descriptions, carried out with program mvec.

dscx → mvec

Then we start focusing on certain parts of the image by using processes such as focus selection and shape description, excecuted with programs focsel and shpx, resp. Those two processes can be regarded as attentional shifts. The output of focus selection can be matched again with program mvec:

focsel → mvec

The output of shape description is matched with program mshp:

shpx → mshp

We elaborate on the individual processes:

- dscx [descriptor extraction]: the binary outputs the features and descriptors into a number of different files of which the three main ones are:
  - the *vector* file with extension .vec, containing the descriptor attributes with which one can span a multi-dimensional space, useful for identification.
  - the *histogram* file with extension .hst, expressing the attributes as histograms, useful for fast classification with a 'traditional' classifier (Linear Discriminant Analysis [LDA], SVM, RandomForest [RF], etc.).
  - the *saliency* file with extension .slc, containing scene statistics, object proposals and texture information (Section 2.2.3). This information can be used to decide where to apply focus selection and shape description. And it can be used for visual orienting, for example for auto-focusing, zooming, saccading (change of camera direction), etc.

- mvec [matching vectors]: matches the descriptors as outputted by the program dscx or focsel, and returns metric measurements for various types of descriptors. This is useful for identification of structure. The use of this binary will be explained in Chapter 3.

- focsel [focus selection]: extracts descriptor vectors of a desired region, a so-called *focus*, from the vector file as generated by dscx. The region is defined by the user as a bounding box and can outline an object proposal or part proposal,

ie. obtained from the saliency file; or it can be an annotation. The bounding box can be of arbitrary dimension and size. `focsel` extracts both, vectors and histograms, extensions `.vef` and `.hsf`, resp. The vectors can be matched with other focii with the program `mvec`. To be further detailed in Chapter 5.

- `shpx`/`mshp` [shape extraction and matching]: refines the segmentation of a shape and saves it to file with extension `.shp`, which then can be used for matching (Chapters 6 and 7). This shape can be any silhouette, be it the letter of some text in the wild, or an object with homogenous color, or scene part.

With that program survey, we can now refine the above pipeline. Since the deployment of vector matching - using `mvec` - is a relatively costly process, it makes sense to subselect candidates by firstly classifying the histogram output, process *Clsf*, and then to apply `mvec` on the identified subset of representations:

`dscx` → *Clsf*(`.hst`) → `mvec`(`.vec`)

This multi-stage (cascaded) process can be based on applying a classifier (LDA, SVM, RF, etc.), which makes sense if we assume a clearly defined category. Or it can be based on histogram matching only, if the goal is to identify a structure. For the former there exists enough software; for the latter we provide a separate program called `mhst` (Chapter 4).

Focus selection can be based on the output provided by the saliency file, or any output of `dscx`. For matching, we can again make a selection based on classification:

`focsel`(`.slc` | `.vec`) → *Clsf*(`.hsf`) → `mvec`(`.vef`)

Focus selection allows applying tailored representations, without performing a complete feature extraction and description. It is useful in particular, if we wish to analyze structure containing contour information or texture. If the goal is to analyze rather a shape silhouette in more detail, then we apply shape description and matching:

`shpx`(`.slc`) → `mshp`

Or one can apply both processes to the same patch. We will refine the pipelines even more, after we have introduced the full set of programs (Section 1.8). Next we illustrate the usage of those programs.

1.2.2 Introduction to Usage

The following examples give an idea of how to provide the arguments to the program binaries.

The task is to compare two images named `imgA.jpg` and `imgB.jpg`. Firstly, we generate the descriptors and provide a filename as output, in this case single letters `A` and `B` (both in directory `/Desc`):

```
> dscx imgA.jpg Desc/A
> dscx imgB.jpg Desc/B
```

This will write the vector files called `A.vec` and `B.vec` to directory `/Desc`. In a first round we compare the images at once, for which we feed the vector files as arguments

to program `mvec`,

```
> mvec Desc/A.vec Desc/B.vec
```

which returns dissimilarity and similarity metrics, either as standard out (`stdout`) or as file.

In a second round, we compare two different regions, for which we now deploy `focsel`. We select the upper left quadrant as bounding box, `0 128 0 128` (assuming image sizes are both 256x256):

```
> focsel dscA.vec 0 128 0 128 focAupplef
```

This will write the subset of descriptors to file `focAupplef.vef`. We extract an equally sized region from image B and call it `focBsomewhere` (operation not formulated here). Then we match those two regions:

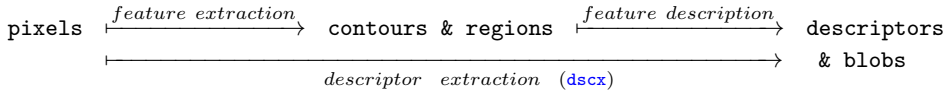
```
> mvec focAupplef.vef focBsomewhere.vef
```

which again returns the metric measurements. We integrate the results ad libitum.

The programs allow to take lists as input, ie. one can specify a text file as input that contains file names or bounding boxes. The details of that will follow in the chapters.

1.3 Structural Description

The structural description is based on contours and regions. The extraction of those is called *feature extraction*, resulting in lists of contours and regions (Section 1.3.2). Those features are then partitioned, parameterized and integrated, which is referred to as *feature description*. The output of feature description consists of so-called *descriptors* as well as texture blobs, explained in Sections 1.3.3 and 1.3.4. The entire extraction and description process is called *descriptor extraction*, hence the program name `dscx`. Program `dscx` outputs the descriptors and texture blobs, but also bare region information.



The outputted description is quite rich and can be deployed differently, ie. selected and interpreted for a specific task (Section 1.3.5). We now firstly explain from what architecture the features are extracted from.

1.3.1 Architecture

Contour and region features are extracted from an entire image pyramid with `nLev` levels, depicted here for `nLev=4`, using zero-indexing:

```
lev 3        --
lev 2       ----
lev 1      --------
lev 0  ----------------
```

The bottom level of the pyramid, `lev 0`, holds the original image resolution. Higher pyramid levels are generated by downsampling with an integer factor equal two. Downsampling continues until the map is equal 16 pixels; or just larger, for the smaller side length. In the depicted schema, level equal 3 would be the top level and would correspond to the pyramid of a 128x128 pixel image. The number of levels can be set as command argument using double dash to specify a long option:

```
> dscx imgA.jpg /dscA --nLev 2
```

The parameter can also be set by file, the details of that follow in later sections. For each level of the pyramid, contours and regions are extracted, which will result in a vast set of features. For efficient recognition, some sort of selection must take place. We firstly introduce feature extraction and its selection parameters (Section 1.3.2), followed by the descriptors developed and their selection procedures (Section 1.3.3).

There are no particular image preprocessing algorithms carried out in this program, as I have never observed any consistent improvement of using for instance smoothing or low-pass filtering. In fact, I have the impression it is detrimental sometimes. But there may be tasks or databases where filtering could be useful. In that case, it is best to carry out the preprocessing separately (beforehand) and then feed the preprocessed image to `dscx`.

The pyramid architecture is suitable for the fast analysis of arbitrary image content. If we know what type of input we face, then a scale space may be better suited, in particular when subtle differences need to be discriminated. To be offered in future implementations.

1.3.2 Feature Extraction

Contours `cnt, rre`

Three types of contours are extracted, namely ridge, river and edge contours, sometimes abbreviated as `RRE` or `rre`. They are kept separately initially and we depict that as three separate pyramids. We use symbols . and | to symbolically express that the pyramid content can be quite individual depending on the image, meaning the levels have different list lengths with different segments lengths (no actual correspondence is depicted with those symbols):

```
               ridges                rivers                edges

    lev 3        |.                    ||                    ..
    lev 2       ||.|                  |.||                   ||||
    lev 1      ||.||||              ||||||.|              |.||||||
    lev 0  |||.|||||||.||.||      ||||.|.|||||||||       ||||||||||..||||
```

The threshold parameter for accepting a pixel value as a contour pixel is set with `Cnt.minCtr` by file or `--cntMinCtr` by long option. By default the value is 0.05 and is relative to the maximum value of the range image (taken with a 3x3 neighborhood).

Regions `reg`

Regions are detected by a hierarchical thresholding process, whose tree `depth` is typically set to value equal three for images up to ca. 400 x 500 pixels.

The thresholding process is applied to each level. The following illustrates the map output for an architecture with four levels and depth equal three:

```
            depth 0           depth 1           depth 2
lev 3         --                --                --
lev 2        ----              ----              ----
lev 1      --------          --------          --------
lev 0  ----------------  ----------------  ----------------
```

The creation of small regions can be controlled by parameter `Reg.minPixNode` (or `--regMinPixNode` by long option), more details in Section 2.3.3. There is no contrast threshold applied here. The boundary pixels are saved to a file with extension `BonPix` (Section 2.2.6).

The folder `/DemoBaum` contains programs that demonstrate the output for the original resolution, that is `lev=0`, see Section 1.7.

Each of those maps is then analyzed for connected components and their boundaries are extracted. The boundaries are concatenated across depth (per level), resulting in a pyramid structure as depicted already above for contours. The boundaries themselves will be used for a simple description, called *radial-shape* descriptor or sometimes just radial descriptor. Boundaries are also partitioned into curved segments (arcs) and straighter segments (straighters).

The divisive region segmentation process is capable of returning regions of lowest contrast possible. This sensitivity results in a vast feature output that is useful for search scenarios, such as defect detection of material surfaces, anomaly detection, detection of occluded objects or of low-contrast objects, etc. But for most recognition tasks it requires some sort of selection, based on parameters such as contrast, minimum size, minimum spacing, etc. Some of this selection is better performed on the parameterized features, the descriptors.

The default parameters for selection are set such, that one can perform decent classification and identification of the images as appearing in image collections with daily scenes. For specific tasks one had to adjust some of the parameters, which will be discussed throughout the explanations and be subject when discussing applications (Chapter 9).

1.3.3 Descriptors

The parameteric description is typically carried out for all features, and the resulting lists of descriptors are also called the *full* set. When we subselect features for a specific task, then the resulting subset is called *skeleton* in our publications, but we use here another term as well, for clarity. We here mention only the key parameters for this selection, as this selection has the largest influence on classification accuracy. The complete list of parametric options will be provided in later sections, as well as the individual attributes. There are currently 7 descriptors, of which the following four are considered basic descriptor types:

- **Contour** (`cnt`) : describes the ridge, river and edge contours (RRE). An individual contour is described by its length and angular orientation. The use of all three types constitute the full set, also called *RRE set*. Based on this RRE set, we derive three types of descriptions:

  - Skeleton: a selected set of longer contours, called *skeleton* sometimes. The skeleton is saved to the vector file and appears as `ACNT` when loading (a list of pyramid levels).
  - Bundle: groups of contours, to be further introduced below.
  - Texture: a texture analysis based on the orientation angle of segments, to be introduced in Section 1.3.4.

  The RRE set is *not* saved by default, due to their size. It can be saved by turning on long option `--saveRRE`, in which case it is saved to a separate file with extension `vecRRE`.

  The reduction from the RRE set to the skeleton set occurs by a global-to-local selection procedure, whose parameters are minimum spacing and minimum contour length, see `sklMinSpc` and `sklMinLen` of Section 2.3.2.

- **Radial shape** (`rsg`) : describes the region boundary using a radial signature. The description occurs for boundaries of minimum contrast and minimum size, see `rsgMinPix` and `rsgMinCtr` of Section 2.3.4.

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

  The full set of arc segments is loaded to `AARCfll` by routine `LoadDescVect`. The selected set is called *gerüst* (scaffold) and loaded as `AARCgst`. The parameters for selection are analogous to the ones for contour selection.

- **Straighter segment** (`str`) : describes a straighter boundary segment obtained from partitioning the region boundaries. The straighter segments are essentially the segments lying between arc segments. Analogous to arcs, the full and selected set are called `ASTRfll` and `ASTRgst`, respectively.

These basic descriptors alone provide relatively good categorization and identification performance. But for interpreting our environment more efficiently, we form more complex descriptions:

- **Shape abstraction** (`shp`) : a description based on the segment statistics of arcs and straighters for each individual shape. It contains dozens of attributes. By default, this shape description takes place only for regions that are fully inside the image, that is not touching any image border. There exist scenes that lack any inside shapes, such as landscape scenes, photos of smooth surfaces, etc. In that case, it might be worthwhile including shapes that touch the image border. This can be regulated with parameter `bordTouches`. By default it is set to 0 and ignores any regions touching an image border. With value equal 1, a shape can touch one image side; with value equal 2, two sides; with value equal 3 three sides; and with value equal 4 all four sides.

- **Tetragon descriptor** (`ttg`) : a description focusing on shapes that contain at least two loosely parallel straighter segments. It is a subset of the shape descriptors (`shp`), but with a more refined parameterization.

- **Bundle descriptor** (`bnd`) : a description based on groups of RRE contours. Groups are detected when determining the skeleton set and therefore show the minimum length specified by `sklMinLen`. Groups of shorter segments are better expressed with the texture analysis.

The program `dscx` outputs all lists of descriptor vectors to a `vec` file, placed into directory `/Desc` in our code examples. This is summarized in Section 2.2.1. It also outputs them as bins to a `veb` file for building histograms.

The descriptors come with position and angular attributes. The position is often the (normalized) coordinates of the center pixel of the feature, or it is calculated from some keypoints. The angle attribute describes the orientation of the feature in the image plane. More explanations on the attributes will follow when introducing the individual programs.

1.3.4 Texture

The texture description is based on the full set of contours, the RRE set. For a small rectangular region of the image, ie. 16x16 pixels, a 4-bin histogram of the contour's angles is formed. Then the histogram is parameterized focusing on its peaks, that is its dominant orientations. The following texture biases are calculated:

- `Num`: total number of contours present in the rectangular region. The parameter can be exploited to detect small objects.
- `Blk`: lack of texture (blank), that is not more than three contour segments are present, ie. sky region, water surface (with no reflection), or any region void of texture.
- `Nil`: degree of 'no' orientation (nil dominance). Often corresponds to foliage.

- `Vrt`: degree of vertical orientation (dominance). Often corresponds to texture in natural scenes, ie. grass, crops, stems, etc.
- `Hor`: horizontal orientation dominance, for example uneven water surface; objects arranged with increasing (spatial) depth, a column of cars.
- `Axi`: axial orientation dominance, that is vertical and horizontal (approximately) equally present, ie. windows in urban scenes.
- `Uni`: single (uniform) orientation dominance. This includes vertical and horizontal dominance, but excludes axial dominance (as there are two orientations present). Can occur at any angle. Rather rare in regular scenes, but potentially useful for textures with diagonal dominance.

The texture description is applied to an overlapping grid of regions, where the overlap is half of the region side. For example for a 16x16 pixel region, the overlap is then 8 pixels, resulting in texture map of size 32 x 32 cells, for a 256x256 pixel image. This process is similar to spatial histogramming of descriptors (upcoming in the next section).

In those texture maps, we determine regions where the above biases are dominant, here called *blobs*. Thus, depending on the scene, we have blobs of different texture types and size. When we load the texture information in Matlab, it requires the following list of strings for proper identification:

```
aBlobTypS = {'Num' 'Blk' 'Nil' 'Vrt' 'Hor' 'Axi' 'Uni'};
```

The variable is instantiated in `globalsSB.m`. The texture information is outputted in the saliency file (Section 2.2.3).

1.3.5 Representation Formats

The descriptor attributes can be deployed in different forms. In the simplest form, we build histograms, suitable for traditional classifiers (LDA, SVM, RF). This fast classification can serve for the subselection of candidates for more specific matching, in short, we build a cascaded recognition process. In the most elaborate form, we use the attributes in a multi-dimensional space, that is then suitable for identification of structure. We start with elaborating on the histograms.

Histogram of Attributes

The attributes are histogrammed individually, with 5 to 12 bins each, both univariate and bivariate. They are concatenated to a single histogram vector. To exploit the position attribute, spatial histograms are formed as well, with both the univariate and bivariate distributions. In total, four types of histograms are generated:

flat, univariate	one-dimensional, taken for the entire image (or focus)
flat, bivariate	two-dimensional, taken for the entire image (or focus)
spatial, univariate	one-dim., taken from a 3x3 grid by default
spatial, bivariate	two-dim., taken from a 3x3 grid by default

They are also referred to as Histogram-of-Attributes. The total dimensionality is currently at ca. $24k$ for a 3x3 grid. Program `dscx` generates all four of them and are

saved in a separate file with extension .hst. Program focsel generates only the flat histograms and they are saved with extension .hsf.

The histograms from different images can be collected with program binary collhimg. They can then be matched with the program mhst, or trained with a traditional classifier. Those programs will be introduced later.

Vector

The vector matching programs mvec and mshp exploit the attributes as a multi-dimensional space. The vector metric uses (in most cases) individual weights for the attributes (one weight per attribute, not for each descriptor). The metric includes the position and angle attribute by default, in which case, the representation can be called a *rigid vector template*. It is useful for identification of structure. Since structure sometimes appears at different orientation or with some variability, one can loosen the template by lowering the weight values, in which case the representation becomes rather a loose (vector) template. If we turn those weight values off (to value equal zero), in particular the ones for position, then the representation is rather a statistical one, namely a set of structural elements. The matching programs allow to control those weight parameters.

1.4 Main Processes

The following summarizes those binaries that carry out computational processes. Most of them were mentioned already.

- dscx: carries out feature extraction and description as introduced already above and outputs various files that will be loaded by other recognition processes. Currently it takes only one image at a time and it is therefore somewhat slow due to repeated memory allocation for individual calls. Future versions will include processing a list of images for faster extraction.
- mvec: matches the descriptor vectors as outputted by dscx, and returns dissimilarity and similarity measurements for each pyramidal layer and descriptor type. It can be applied to any structure: shape, object or scene. There are two instantiations of vector-matching. One that matches only one pair of images (outputs of dscx), called mvec1. And one that matches one versus multiple, called mvecL , 'L' for list.
- mhst: matches descriptor histograms and outputs a dissimilarity measurement. This can be used to identify candidates for vector matching (with mvec). This not only accelerates recognition, it most often improves prediction accuracy.
- shpx: carries out a refined segmentation for a target region using a color cue, and then describes the obtained region silhouette using the radial description (rsg), the arc description (arc) and straighter description (str).
- mshp: matches the description as produced by shpx.
- motvec: computes the motion vectors between descriptors of two frames (Section 3.3). This is essentially the same as mvec1, but outputs in particular the motion vectors between nearest neighbors. That allows to estimate motion flow.
- knnv [prototyped]: nearest-neighbor search of structures using a coarse-to-fine strategy to accelerate the retrieval process, as opposed to mvec above, that

matches the entire pyramid between two structures (and is therefore slower). knnv 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. For the moment one can use the combination of mhst and mvec to accelerate recognition.

1.5 Learning

A learning process will be provided that determines category-characteristic descriptors by individual matching of vectors. In principle one can carry out such a learning procedure with the matching binaries mvec, but it is of course more convenient to have binaries that provide more automation.

- kkcan [planned]: searches for nearest neighbors across images of one category to find candidate descriptors. This is based on mvec1 as introduced above.
- kkgrp [planned]: refines the search and selects a final set of category-characteristic descriptors using some clustering technique.
- kkmtc [planned]: matches the category-characteristic descriptors against a new image and outputs the degree of dis-/similarity per category-characteristic descriptor, similar to mvecL.

1.6 Utility Processes

The following utility processes are available.

- focsel: selects descriptors from a region (focus), specified by the user as bounding box. Program fochst1 extracts the histogram for one focus, program fochstL does so for multiple focii. Histograms are written to files with extention hsf. Progam focvec1 extracts the vectors from one focus; they are placed into a file with extension vef. The vef or hsf files are loaded by matching programs such as mvec, mhst or knnv.
- collhimg: concatenates the attribute histograms of individual hst files into a single matrix of size [nImg nBins], suitable for classification with traditional classifiers (LDA, SVM, RF, etc.), introduced in Section 2.4.
- collvec [prototyped]: concatenates the attribute vectors of a list of vector files to a single matrix of size, [ntDsc nAtts], that is total number of descriptors times number of attributes of a descriptor. This matrix is suitable for clustering, ie. word formation.
- ptchx [prototyped]: Extracts rectangular patches of any size from one image using a list of specified bounding boxes. This serves to prepare the patches for shape extraction with shpx.

1.7 Demonstration, Varia

The following mentions the demonstration programs available, as well as administrative and utility routines for Matlab:

- /DemoBaum contains binaries that demonstrate the segmentation process, one for gray-scale analysis (as used in dscx), and one for a chromatic analysis using the RGB channels.
- /DemoPlcRec: contains Matlab scripts that carry out a simple place recognition experiment using programs dscx, mvec and focsel. It is an elaborate version of the example introduced in Section 1.2.2.
- /DemoSgrRGB: contains a binary called sgrRGB demonstrating the color-segregation process used for shape extraction (Chapter 8). It segregates an RGB image 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. It also outputs the arcs and straighter descriptors for the foreground regions. More information in Chapter 8.
- /UtilMb: contains utility routines for Matlab.
- /UtilPy: contains utility routines for Python. In development.

1.8 Recognition Pipeline

We have introduced a simplified pipeline already previously. With the full set of programs introduced above, we can now envision full pipelines for different tasks. We start with a generic recognition pipeline, and then suggest one for the task of text recognition in the wild.

In case of abscence of a specific task, recognition may start with scene classification (*ScnClsf*), that allows to select candidates for scene identification (*ScnIdfc*). Then one proceeds with a more local analysis, such as object or scene part recognition (*ObjClsf*, *ObjIdfc*), followed by a shape analysis:

`dscx` → *ScnClsf*(`.hst`) → *ScnIdfc*(`.vec`, `mvecL`) → `fochst1`(`.slc`) → *ObjClsf*(`.hsf`) → *ObjIdfc*(`.vef`, `mvecL`) → `shpx` → `mshp`

For the classification processes (*Clsf*) we have not named an algorithm, since there exists a large body of classifers. If the goal is scene identification only, then one could deploy `mhst` to subselect candidates. If the goal is classification per se, then one can obtain decent results with the combination of PCA and LDA; a Random Forest classifier can sometimes achieve substantially better accuracies and is carried out much quicker during application. One could also try feature selection schemes.

The saliency file provides generic object proposals, but for specific tasks it is of course more fruitful to create proposals based on the descriptors themselves.

For text recognition in the wild, the above generic pipeline is useful to generate approximate candidate locations. We here focus on the part of the pipeline, that attempts to find letter shapes. In that case it makes sense to classify individual shapes immediately, in order to locate character and text candidates as quickly as possible. For that we extract a focus at those shape locations (proposals), that are provided by the saliency file (`.slc`). We can deploy shape extraction and matching (`shpx` and `mshp`) to discriminate between letters and non-letters.

To achieve the highest identification accuracy possible, we apply the (demo) program `sgrRGB`, which returns the shape boundary, that in turn can be applied to an OCR network:

`dscx` → `fochstL`(`.slc`) → *Clsf*(`.hsf`) → `shpx` → `mshp` → `sgrRGB`(`.slc`) → *OCR*

Chapter 2

Descriptor Extraction [/DescExtr]

The program dscx outputs a number of files, that are written to directory /Desc in our examples. The files contain the parameteric description; the bounding boxes for salient regions; the keypoints of contour segments; the histograms of attributes (for fast classification); and files containing boundary information. The directories contain the following:

/Desc	output directory for description
/Imgs	sample images for immediate probing
/Params	contains example files for setting parameters
/UtilMb	Matlab routines and example scripts to read the output data files
/UtilPy	Python scripts to read the output data files. In development.

The use of the program is explained first (Section 2.1), followed by explaining what type of data files it generates (Section 2.2). Then we introduce the available flags and options (Section 2.3). Finally, we explain how to collect the histogram output of multiple files, using program binary collhimg, to generate a matrix suitable for training classifiers (Section 2.4).

2.1 Program Use [dscx]

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

```
> dscx pathImg pathOutFile
```

The input image can be jpg or png. The output path must include a slash (as 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.vec`	descriptor vectors (attributes), used by `mvec`
`img1.hst`	descriptor histograms, used by `mhst`, collected with `collhimg`
`img1.slc`	saliency information.
`img1.veb`	descriptor bins, used by `fochst1`/`fochstL`.
`img1.itgc`	integrated contours.
`img1.utz`	utilities.
`img1.vecRRE`	full set of ridge, river and edge attributes, used by `motvec`
`img1.CntEpt`	only the endpoints of ridge, river and edge segments.
`img1.Bbox`	bounding boxes of regions.
`img1.BonBbox`	bounding boxes concatenated across depth.
`img1.BonAsp`	aspects of boundaries
`img1.BonPix`	boundary pixels

The Matlab script `exsbDsc1.m` in the main directory shows how to execute the program from Matlab, and plots all descriptors using routines as described in later sections. An example for a wrapper routine is given with script `RennDscx.m` that also verifies proper termination of the program.

This is a huge, novel framework tested on ca. 15 databases (tens of thousand of images), yet it might still fail with unusual images. An image larger than 480x640 pixels, that contains fine texture, might exceed some of the allocated memory as I operate with constant allocation and try to reduce memory allocation for large images. Perhaps there exists a natural scene image (>480x640 pixels) with noise-like texture, that exceeds the allocation of contours. If you happen to encounter one, inform me on `info@sehbau.com`. The descriptor extraction program 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 interval checks everywhere.

2.2 Output Data Files

The directory `/DescExtr/UtilMb` contains Matlab function scripts to load the data files and to display the features. Example scripts are given in directory `/Examples`:

`exsbPlotVect.m`	descriptor attributes, function scripts in `/Vect`
`exsbPlotHist.m`	histograms, function scripts in `/Hist`
`exsbPlotShape.m`	illustrates shape abstraction
`exsbPlotTtrg.m`	illustrates the tetragon-like shape abstraction
`exsbPlotBbox.m`	bounding boxes, routines in `/Bbox`
`exsbPlotBon.m`	region boundaries, functions in `/Bound`
`exsbPlotBonPix.m`	function scripts in `/Bound`
`exsbPlotSalc.m`	saliency information

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

2.2.1 Vectors (.vec)

The vector file with extension .vec contains the descriptor attributes and can be loaded with routine LoadDescVect.m,

```
[DVEC Kt]   = LoadDescVect(pthVect);
```

Structure DVEC contains the descriptor pyramids:

.ACNT	skeleton of contours (not full RRE set)
.ARSG	radial description of region boundaries
.AARCfll	full set of arc segments ('fll' for full)
.ASTRfll	full set of straighter segments
.AARCgst	selected set of arc segments ('gst' for gerüst)
.ASTRgst	selected set of straighter segments
.ASHP	shapes (typically not touching borders)
.ATTRG	tetragons, preciser shape info of elongated shapes
.ABNDG	bundles, clusters of contours

which are loaded with a routine Read[Dsc]Spc.m, that in turn calls a routine Read[Dsc]Att.m ie. ReadCntSpc.m and ReadCntAtt.m for contours.

The term 'vector' is in fact slightly misleading, as the attributes are not stored in an actual vector format, but needed to be concatenated to form a vector, explained below.

The output can be manipulated as shown in script exsbPlotVect.m. For the shape descriptors there are the example scripts exsbPlotShape.m and exsbPlotTtrg.m.

The attribute values are organized per type (dimension), not per vector (descriptor), more formally they are struct-of-arrays, not array-of-structs. Thus, for the purpose of clustering or classification in Matlab (or Python), one has to concatenate them horizontally to a [nDsc x nAtt] matrix. Most attribute values are normalized to unit range, some only to approximate unit range in case of complex attribute definitions. Angle values come mostly in radians - or are also scaled to unit range. Depending on the exact type of use of the vectors, one should certainly consider scaling them.

We firstly introduce attributes that are common to most descriptors. Then we introduce the details of the individual descriptor types.

General Attributes

The general attributes include the position of the descriptors in the image (or map), their chromatic values, their contrast, smoothness of the segment, size, angle, etc.

- PosV, PosH: vertical and horizontal position. Typically the center of the feature, ie. the midpoints of contours and boundary segments; the pole for radial descriptors. Read with ReadAttPos.m. These values are normalized to $[0, 1]$, unlike the points provided below.
- RGB: chromatic red-green-blue triplet. Read with ReadAttRgb.m
- Pts: the key-points, such as the two endpoints and the point in-between. Read with ReadDescPtsS.m (short), or ReadDescPtsF (float). Usually in absolute (unscaled) coordinates, corresponding to the pyramid level.

- `Smo, Ter.Smo`: curve smoothness, for arcs and straighters. This is a local measure that expresses the proportion of smoothness in the curve. A L-feature is considered very smooth as it contains only a small proportion of non-smoothness in its corner. The measure is useful to discriminate between natural scene boundaries or boundaries of specular reflections versus boundaries of structure, which tends to be smooth.
- `OrgCrv`: shape/region label. The shape from which the boundary segment was taken (arc or straighter).
- `OrgDth`: the depth map at which the boundary segment was obtained (0-indexing). Perhaps useful in an analysis progressing from high to low contrast.
- `Tif`: holds range information ("Tiefe"), as measured by a depth camera. Not (fully) operational yet. Read with `ReadAttTif.m`.

Other attributes that are common to most (or some) descriptors are:

- `Les` arc length scaled, ie. for contours, arcs and straighters
- `Len` arc length absolute
- `Ori` orientation angle, $\in [\frac{-\pi}{2}, \frac{\pi}{2}]$ (half circle)
- `Dir` directional angle, $\in [0, 2\pi]$ (full circle)
- `Ctr` contrast, normalized or $\in [0, 255]$

In the following we survey the individual descriptor types. Groups of attributes are often loaded with the following routine:

```
[ARR szD] = ReadRackFlt( fid, aFieldNames )
```

which loads the attributes as matrix `ARR` of size `[nDsc nAtt]`, namely number of descriptors times number of attributes.

Contour Attributes `ReadCntAtt.m`

The most valuable attributes are the length and angular orientation as mentioned above already. Another attribute is the degree of straightness `str`, which however shows little relevance for vector matching so far.

Rsg Attributes `ReadRsgAtt.m`

The most valuable attributes are the radius `rds`, the elongation `elo`, the degree of concavity `cncv`, and the circularity `cir`. The other attributes aid classification (with Histogram-of-Attributes from the `hst` file), but have not played a large role yet in vector matching.

Arc Attributes `ReadArcAtt.m`

The geometric parameters are loaded to variable (`.Geo`) as matrix `[nArc nAtt]`, with the above mentioned routine `ReadRackFlt.m`. The first column contains the curvature measure, which is the most valuable attribute. The others improve classification with histograms, but have not played a large role yet in vector matching.

Straighter Attributes `ReadArcAtt.m`

Contains the same attributes as for contours (above), but some other attributes are loaded as well, that are still under development.

Shape Attributes `ReadShpAtt.m`

Shape attributes are organized into several groups of attributes (loaded with `ReadRackFlt.m`). Two groups represent a description of the straighter segments of a shape, and are useful for scene analysis of indoor or outdoor scenes, where there exist often vertical and or horizontal structures. The first group is a coarse description using a 8-bin histogram of the straighter orientations, which then is analyzed for various axial alignments. The second group is a more refined description using a 12-bin histogram. In Matlab those two groups are read as matrices as follows, see function script `ReadShpAtt.m`:

```
[V.STR szL] = ReadRackFlt(fileID );
V.SFI       = ReadRackFlt(fileID );
```

where `.STR` and `.SFI` are coarse and fine information, resp. In the example script `exsbPlotShape`, those matrices are turned into structures by the utility routine `u_AttsArrToStruct.m` and variables `AVEC.LabShpScors` and `AVEC.LabShpScors`, that contain the fieldnames:

```
Scors = u_AttsArrToStruct( SHPlv.STR, AVEC.LabShpScors );
Sfine = u_AttsArrToStruct( SHPlv.SFI, AVEC.LabShpSfine );
```

The variable structure `Scors` has the following fields. Each one represents a degree of a specific structural bias, ranging from 0 (absent) to 1 (fully present).

`Vrt`	verticality
`Hor`	horizontality
`Dg1`	diagonal 1
`Dg2`	diagonal 2
`Axi`	axiality: both vertical and horionzontal
`Adg`	axiality along diagonals
`Vab`	deviation from verticality
`Hab`	deviation from horizontality
`Dab`	deviation from diagonality
`Tri`	three axes dominating
`Nil`	no axis is dominating: shape irregular

This coarse information is useful for abstract classification. The fields of the finer description (structure `Sfine`) are as follows. Some are the same in structural type as listed above, but not in value as we deal with a 12-bin histogram.

Vrt	verticality
Hor	horizontality
Vti	vertical with some inclination
Hti	horizontal with some inclination
Vob	vertical oblique
Hob	horizontal oblique
Dg2	diagonal 2
Dg1	diagonal 1
Axi	axial: both vertical and horizontal
Uni	one orientation
Dul	two dominant orientations (such as in axial)
Cvg	two dominant oris and converging
Agx	angle between the two most dominant orientations
Ori	orientation value of the (most) dominant orientation
Nil	no dominant orientation present
Dre	three dominant oris present
Vir	four dominant oris present
Fnf	five dominant oris present

Tetragon Attributes `ReadTtgAtt.m`

These attributes describe shapes, whose sides appear to form a tetragon that is loosely aligned with either vertical or horizontal axis. The geometric attributes are organized into four groups (as structures of the output variable):

.GEOM	basic form parameters, such as length, elongation, etc.
.LAGE	alignment with respect to the vertical and horizontal axes
.ANGS	angles for converging direction and intersection
.DICV	directional biases

More details will follow.

2.2.2 Histograms (.hst)

The Histogram-of-Attributes for an image is saved to a file with extension `hst`. It can be loaded with routine `LoadDescHist.m`, as demonstrated in example script `exsbDscx1.m`:

```
[HFU HFB HSP Nuni Nbiv Nspa] = LoadDescHist([pthOut '.hst']);
```

where variable structures `HFU`, `HFB` and `HSP` contain the histograms, flat univariate, flat bivariate and spatial (both uni- and bivariate), resp. Variables `Nuni`, `Nbiv` and `Nspa` contain the corresponding bin count.

The histograms of different images can be collected with program `collhimg` to create a matrix for classification (Section 2.4). We recommend starting classification with the first two types, the flat univariate and bivariate histograms. The third type, spatial univariate, improves accuracy in particular for whole images. The last (fourth) type, spatial bivariate, did not consistently improve classification accuracy, when tested on various image collections. It is perhaps best to apply feature-selection schemes.

2.2.3 Saliency (.slc)

The saliency file contains proposals for objects, scene parts and texture blobs, as well as statistical information taken from all descriptors. The proposals were taken from the descriptor information as presented so far, with simple selection algorithms; there is no separate process that identifies new features. Rather, it is an attempt to exploit the vast information obtained with the feature and descriptor extraction process.

The proposals represent a generic selection. For specific tasks, one would create a specific selection procedure. For example if we search for small objects, one would focus on ridge/river/edge contours and omit the large proposals, ie. omit large shapes, see also Section 9.3.

The example script `exsbPlotSalc.m` demonstrates how to read the variables returned by the loading routine `LoadDescSalc.m`:

```
[YZ Shp Ens Dsc] = LoadDescSalc( pthSalc, aBlobTypS );
```

where input variable **aBlobTypS** is a list of texture labels, defined in `globalsSB.m` (Section 1.3.4). The output variables are structures containing the following:

- `YZ`: texture blob information based on contour statistics as explained in Section 1.3.4, which is particularly useful for detecting common scene textures and small objects.
- `Shp`: bounding boxes of salient regions, which are taken from the pyramid of shapes `ASHP`. They are read with reading routine `ReadShpOut`.
- `Ens`: an ensemble of proposals that is combination of the contour (`YZ`) and shape information (`Shp`).
- `Dsc`: statistical information of the descriptors, both the number of descriptors as well as some of their attributes.

In more detail they are:

- `YZ.Blb` this structure contains the blob information, the regions outlining contour texture. It is loaded with reading routine `ReadBlobOut.m` and contains the following structures and fields:

`.Box`	bounding box values in fields `.Top`, `Bot`, `Lef`, `Rit`.
`.Typ`	texture bias, numbering 1 ('Num') to 7 ('Uni'). 8: high contrast
`.Cvg`	coverage of image, $\in [0, 1]$

There may exist multiple blobs per texture type. The bounding boxes for `Typ=1` are the most general ones and outline any blob containing contour segments: if the boxes are small, than they outline small objects in isolation.

- `YZ.Spt` contains a selection of points that represent clusters of high contour count and of high contrast, relative to their immediate context. They are plotted in the section called 'Spots' (in `exsbPlotSalc.m`), with routine `p_VisSearch`. These points are useful if the scene contains spots of moderate cluttering that we wish to locate, ie. when searching for objects.

- `Shp` this structure summarizes key aspects of the selected shapes:

`.Box`	bounding box values in fields `.Top`, `Bot`, `Lef`, `Rit`.
(`.Typ`	irrelevant (NOT texture bias))
`.Cvg`	coverage of image, $\in [0, 1]$
`.Ctr`	contrast, $\in [0, 255]$
`.Cwd`	crowdedness, similar to texture bias `'Num'`
`.Lev`	pyramid level from which a shape was taken, $\in [0, \texttt{nLev} - 1]$
`.IxShp`	index to shape for the given pyramid level (in `Lev`)
`.IxBon`	index to boundary (level in `Lev`)

- `Ens` the structure for ensembles contains:

`.Box`	bounding box values in fields `.Top`, `Bot`, `Lef`, `Rit`.
`.Typ`	descriptor type: 1-8 = contour texture; 10 = shape.
`.Cvg`	coverage of image, $\in [0, 1]$
`.Ctr`	contrast, $\in [0, 255]$. For contours set arbit. to 100. For shapes real value.
`.OrdGtoL`	order of indices from large to small (global-to-local)

- `Dsc.MxRngRR`, `Dsc.MxRngEg` These two arrays contain the maximum contrast (range) value of ridges and rivers (RR) and edges (Eg) per pyramid level. This information is useful for autofocusing; if the RR values are extremely low (less than 10), then we likely face a blank visual field with perhaps specular reflections; or a scene with no clear contours, such as underwater scenes, or with regions of low contrast.

- `Dsc.MaxSizScl` contains the maximum size per level for each descriptor. If the one for shapes is large, ie. `max( MaxSizScl.Shp )`, then we likely have a large object in the image center (if parameter `Shp.bordTouches` = 0, which is default); or it can be background, that is surrounded by objects, such that the background appears as an inside shape.

- `Dsc.GryMmm` is a 3-value array that contains the gray-level value for the pixel with lowest value (`GryMmm[0]`), the mean pixel value (2nd entry), and the pixel with the maximal value (third entry).

The saliency file does not hold any specific proposals from long contours or their groups. For that it is best to immediately use the pyramid of skeletons `ACNT` or bundles `ABNDG` of the vector file. Each descriptor of those pyramids is a proposal.

2.2.4 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 tasks.

The information in this file is minimal. More information is provided in the `.BonXXX` files (upcoming in Section 2.2.6). But here we introduce the organization of the bounding boxes as well as the border values.

The first two integer values of the file hold the number of levels `nLev` and segmentation depth `depth` applied in 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, which is included in the `.BonXXX` files, upcoming in Section 2.2.6.

Selecting the inside regions (0 touches / off border) returns the objects in a scene - if there are any. In landscape scenes (without any large objects in focus), such inside regions can be rare, as the scene parts often touch one or more image sides. Small objects rather appear as ridge or river contours.

2.2.5 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.

2.2.6 Boundary Information (`.BonBbox`, `.BonAsp`, `.BonPix`)

The full boundary information is saved to three separate files.

- `BonBbox`: bounding box information, similar to `Bbox` (as introduced above), but in different format.
- `BonAsp`: more boundary aspects.
- `BonPix`: boundary pixels.

The two scripts `exsbPlotBon.m` and `exsbPlotBonPix.m` demonstrate how to load those data files. The file with extension `BonBbox` contains essentially the same information as the `Bbox` file (described above under Section 2.2.4), but in slightly different format and with additional information. The differences are:

- the bounding boxes are concatenated across depth, but the depth information is still available as the 6th parameter.
- the box coordinates are scaled to original size already.
- the contrast value for the boundaries is given as 5th parameter.

The file with extension `BonAsp` lists some additional boundary aspects:

- chromatic values red, green, blue for the pixels along the boundary, not the region inside. Thus for small regions it may not be an optimal chromatic representation.
- coverage of the boundary area, as proportion of the image/map
- border values as introduced under Section 2.2.4 already.
- perimeter, which is given as absolute value.
- area in pixels for the connected component (not the boundary itself), thus excluding holes
- area of boundary and thus including holes.

The `BonPix` file is loaded with routine `LoadBonPix.m`.

2.3 Options and Parameters

Options and parameters can be passed either as text file or as long options. For the use of a text file see the directory /Params. The file named PrmDesc_Example.txt contains the most important parameters. If one uses both, long options and text file, then the text file must appear as third argument, followed by long option arguments:

```
> dscx Imgs/img1.jpg  Desc/img1 Params/PrmDesc_Example.txt --depth 4
```

whereby the values provided by the long options overwrite those provided by the text file. The naming of parameters is slightly different for the two types occasionally. On the long term, the naming as read from file will be the preferred one, as it is more systematic.

Many of the parameters will mainly regulate the number of descriptor vectors saved to the vec file. For histograms, typically the full set of descriptors is taken (rre for contours, full for curve partitions), as the exact choice of those parameters has lesser influence on recognition accuracy. 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.

In the following the parameter name for the text file is listed first, if existent already. If a long option is present, it is specified with a double dash -- and is listed in parentheses (if the parameter can also be specified in the text file); single letter options are not in use.

2.3.1 Architecture

nLev (--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.

2.3.2 Contours

Cnt.minCtr (--cntMinCtr): contrast threshold for contours. Default =0.05. This is the proportion, of the largest difference found in the range map of the gray-scale intensity image (for a 3x3 neighborhood).

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 flag plot] or used the example script exampleLoadVect.m.

--sklMinSpc: minimum spacing. Default =0.05. This is as proportion of the image side length $\in$ [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 $\in$ [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.

2.3.3 Regions

depth (--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. A depth of five is the maximum.

Reg.minPixNode (--regMinPixNode): minimum number of pixels for a region to be segregated by the thresholding mechanism. This will affect the region count from the second segmentation map on. It will not affect the first segmentation map, as its regions are the first nodes of the tree.

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

2.3.4 Radial Shape

--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.

2.3.5 Partitioning (Arcs/Straighters)

Parameter names starting with cvp regulate the partitioning 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 straighter segments only, as those are rarer. For higher values, it will also start omitting arc segments.

Cvp.bInclBord (--cvpInclBord) [flag]: includes partitions at borders. The default is OFF, meaning that partitions at image borders are ignored. Turning them ON (by listing this option) can improve place recognition significantly.

Cvp.minCtr (--cvpMinCtr): minimum boundary contrast entering the boundary partitioning process. This is a proportion of the largest boundary contrast found in

boundaries. Default equal 0.05.

Arcs and Straighters

Cvp.prpMinLenArc (--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 a fixed threshold $\in$ [0..1] based on the measure chord length divided by segment arc length. Default equal 0.8.

The following options set parameters for selecting the skeleton of partition segments, called here 'Gerust', abbreviated gst/Gst. Reducing the number of segments for finer scales often improves recognition - and also reduces matching duration. But I suspect that for smaller images (or higher levels of the pyramid; ie. smaller 100 pixels per side), that the selection might have less effect.

--arcGstMinSmo: minimum smoothness for arcs set for entire pyramid. Default approximately 0.20. This parameter is intended to eliminate segments (resulting from segregation of luminance gradients) that typically show high irregularity in their spatial course. Images of scenes taken at night tend to have those in particular.

--arcGstMinSpc: minimum spacing for arcs set for entire pyramid. Default approximately 0.05.

--arcGstOff: turns off any selection by setting all parameters values to zero (for smoothness, spacing and length). This then takes the full set of extracted arcs.

--strGstMinSpc: minimum spacing for straighters set for entire pyramid. Default approximately 0.05.

--strGstOff: turns off any selection by setting all values to zero. This then takes the full set of extracted straighters.

2.3.6 Shape (Arcs & Strs)

Shp.bordTouches: number of border touches permitted for a shape. Default equals zero, meaning only inside shapes are described. Values 1 to 4 permit the corresponding number of border touches for a shape.

(Shp.minCtr: minimum contrast for a shape. Not in use yet. Currently depends on the boundary contrasts measured.)

2.3.7 Utility

--prms [flag]: displays the parameter values used. Default OFF.

--noBbox [flag]: turns off saving of bounding boxes (.Bbox file) and contour endpoints (.CntEpt). Default ON.

--noBon [flag]: turns off saving of boundary information (files .BonBbox, .BonAsp and .BonPix). Default ON.

--noBin [flag]: turns off saving of descriptor bins (file .veb). Default ON.

--plot [flag]: plots contours and region boundaries for the entire pyramid. Default OFF. The following images will be written:

-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 [flag]: for illustration or for tracking errors. Default OFF.

2.4 Collecting Histograms [collhimg]

After one has described multiple images with program dscx, one can collect the generated histogram files (hst) to a single matrix with program collhimg. The matrix is of size, nImg x nBin, namely number of image histograms times number of bins (over 24 thousand for 3x3 spatial histogramming). It can be deployed for training traditional classifiers such as LDA, SVM, Random Forests, etc.

The program takes two arguments. The first argument is a text file containing a list of histogram files. The second argument is an output name for the matrix:

```
> collhimg ListHists.txt COLLHST
```

This will write the matrix into the file named COLLHIST.hstc, where letter 'c' in the file extension stands for collection. The matrix can be loaded with LoadCollHist.m. An example script is in directory /MtchHst, called exsbCollHist.m.

The second output variable of LoadCollHist.m, here called Nbin, contains the bin numbers of the individual histograms. The 4-element array Nbin.Tot holds the bin numbers for the four types of histograms, flat univariate, flat bivariate and the two spatial versions.

Chapter 3

Matching Vectors [/MtchVec]

The use of the program for vector matching is explained, called `mvec`. It matches the descriptor vectors of two images (`.vec` files) as generated by the descriptor extraction program (`dscx`), or two focus files (`.vef`) as generated by `focsel`. It can be used for any two structures expressed by the vectors, be it a scene, an object, a shape or a texture. The directories in the folder contain the following:

/Desc	vector or focus files (as outputted by dscx/focsel)
/Imgs	sample images for immediate probing
/Mes	results of matching metrics
/Params	contains example files for setting parameters
/Regist	textfiles containing a list of filenames of vector files
/UtilMb	Matlab routines

The program `mvec` matches the two list of descriptors using pairwise measurements and choosing the nearest neighbor. Two metric measures are available, a dissimilarity and a similarity value, abbreviated `dis` and `sim`, or sometimes also abbreviated as `dist` and `simi`, resp.:

- `dis` (`dist`) returns the Euclidean distance.
- `sim` (`simi`) returns the proportion of matches that are below a fixed threshold value, set with option `[dsc]TolMtc` or `tolMtc` for all descriptor types.

The program can be applied to two vector files (`vec`) as outputted by `dscx`, or to two vector focus files (`vef`) as outputted by `focsel`. The combination of a `vec` and `vef` file is not possible yet. The vector files are required to have the same number of levels (pyramid height), otherwise the program returns no results (more flexibility to be included in the future).

The program comes in two variants:

- `mvec1` : matches one pair of images (or focii): one versus one. Its output is very elaborate, for example it generates nearest-neighbor information suitable for learning category-characteristic descriptors. The Matlab script `exsbFrames.m` demonstrates how to deploy the program and read its output. This program is useful for exploring parameter settings for different levels of the pyramid.
- `mvecL` : matches a list of images, or a list of focii: one versus multiple. Its use is demonstrated in Matlab script `exsbMvecLimg.m`. It outputs the measurements

per descriptor only and not for the entire pyramid (as in mvec1). It is useful for matching at large scale.

Then there exists also the binary motvec, that calculates the motion vectors between nearest neighbors. It is similar to mvec1, but returns the vectors only, useful for estimating motion flow, introduced in Section 3.3.

3.1 Program Use [mvec1, mvecL]

We firstly explain the use of mvec1, the matching of two vector files. Their file paths are given as arguments. For example for two vector files from images (.vec from dscx) we write:

```
> mvec1 Desc/img1.vec Desc/img2.vec
```

Or for two focus file (.vef from focvec1):

```
> mvec1 Desc/foc1.vef Desc/foc2.vef
```

The vector files are required to have the same number of levels, ie. generated by similarly sized images, otherwise it returns no results. The output will be further explained in Section 3.2.

For the use of the program mvecL, we specify a file path as well as a text file that contains the file paths for a list of files to be matched with. We prefer to keep those text files in a folder called /Regist (for register):

```
> mvecL Desc/img1.vec Regist/FinasImg.txt
```

This will write the metric measurements into a file named Vec.txt in directory /Mes. You can specify a different file name by giving a third argument, ie.:

```
> mvecL Desc/img1.vec Regist/FinasImg.txt Mes/ImgVec.txt
```

By default, the program matches all descriptor types for the entire pyramid. The options allow to subselect descriptors and levels, as well as to set attribute weight values.

3.1.1 Options

Options can be set by file or by long options (as in case of dscx). Again, the use of a file is explained through the Matlab scripts. In the following the use of the long options is explained.

The first list of options set a parameter value to the same value for all descriptor types (Section 3.1.1), which can be unspecific in some cases, but which is convenient for coarse tuning. The second list of options allows to adjust parameters of individual descriptor types for fine tuning (Section 3.1.1).

General (All Descriptor Types)

The following options set values for all descriptor types simultaneously and are useful for coarse tuning. Default values are given in approximate values only, as they are often individual to the descriptor type:

--tolMtc [similarity metric]: sets matching tolerance to a fixed value, arcoss all pyramidal levels (and across all descriptor types). This is for the similarity metric only (section `Simi` in output). Default: ca. 0.05.

--wgtRGB: sets the weight value for the RGB difference. Set this parameter to zero if chromatic information is irrelevant, for example when places are to be recognized at either day or night. Keep in mind that three difference values are taken (R,G,B) and that this weight parameter therefore has more influence than the others. Default: ca. 1.0.

--wgtPos: sets the weight value for the position parameter for each descriptor type. A value of 0 turns off the influence. Default equal ca. 1.0.

Individual (Per Descriptor Type)

--cntTolMtc: tolerance for contour matches, for the similarity metric. Fixed value for all levels, but will try to provide something more flexible. Default: complicated.

--rsgTolMtc: tolerance for radial descriptor matches. Analogous to option cntTolMtc.

--arcTolMtc: tolerance for arc segment matches (see cntTolMtc).

--strTolMtc: tolerance for straighter segment matches (see cntTolMtc).

More in progress.

Utility

--prms: displays the parameter values used.

3.2 Output

3.2.1 Program mvec1

Two types of results are returned. One type are the measurements of the list-matching metrics, written to `stdout`. The second type are the nearest neighbor indices (Section 3.2.1). See the example `exsbFrames.m` for the upcoming explanations.

Descriptor List

The list-matching measurements (appearing in stdout) are returned once for the entire pyramid, that is for each level (and each descriptor); and once integrated, for each descriptor type; as well as for the total, called image measure. The values per descriptor type and per image look as follows:

```
-----  desctypes  -----
dty  dis       sim
skl  1.357370  0.117928
rsg  1.480588  0.098934
arc  1.489363  0.009344
str  1.351547  0.054755
shp  5.827399  0.048974
eodty.
-----  img  -----
dis 23.574306
sim 0.000000
eoim.
```

The strings `----- desctypes -----` and `----- img -----` help locating the beginning of the respective measurements sets; as well as the 'end-of' strings `eodty` and `eoim`. This is carried out with the Matlab script u_MtrMesSecs. The actual measurement values are read by routine u_MtrMesScnf.

Nearest Neighbors

The nearest-neighbor indices are written to files in directory /Mes with prefix NNspc and suffix 12, the latter indicating direction of comparison. The indices are kept separate for each descriptor type, ie. NNspcCnt12, NNspcShp12, etc. They are loaded with scripts LoadNNDspace and ReadDescNNs. The section 'Correspondence' in script exsbFrames.m establishes a visual correspondence for illustration.

3.2.2 Program mvecL

The results are written to three files into directory /Mes, called Vec.txt, MesDtyDis.txt and MesDtySim.txt. Each row corresponds to one pair of matching.

- Vec.txt contains the metric measures for the ensemble of descriptor types. It consists of four columns, where the first is the dissimilarity value, the second the similarity value. The third and fourth column are empty (zero) for the moment. The Matlab routine LoadMtchMes.m shows how to load the file.

- MesDtyDis.txt contains the dissimilarity value for each descriptor type. Each one consists of 7 columns, corresponding to contour, radial shape, arc, straighter, shape, tetragon and bundles of contours.

- MesDtySim.txt contains the similarity value for each descriptor. Its organization is the same as for dissimilarity values.

With the output of files MesDtyDis.txt and MesDtySim.txt one can develop individual ensemble measures.

3.3 Motion Vectors [motvec]

The program motvec takes as input two vector files and the arguments are therefore specified analogous to program mvec1:

```
> motvec Desc/frm1.vec Desc/frm2.vec
```

This will output the vectors into directory /Mes in file A.motvec, which can be read as demonstrated with script LoadMotVec.m. By default it will use the skeleton contours as provided in those vector files (pyramid in ACNT). The program will also search for the presence of the vecRRE files, and if they are present, then the RRE vectors are taken for motion calculation, and the skeleton descriptors are ignored.

Since the program dscx does not save the RRE set by default, this has to be triggered by the corresponding flag. Example script exsbMotVec.m demonstrates how to do that, ie. by setting OptK.saveRRE = 1;.

Chapter 4

Matching Histograms [/MtchHst]

The program mhstL matches an attribute histogram against a list of other attribute histograms. This coarse comparison allows to identify a selection of candidates, that are then applied to a classifier, or that are immediately used for more accurate matching with mvec.

The program takes as input a histogram file (.hst file) as generated by the descriptor extraction program dscx, or as generated by focus selection program, fochst1 (or fochstL). The directories in the folder contain the following:

/Desc	histogram files (as outputted by dscx/focsel)
/Mes	results of distance measurements
/Regist	textfiles containing a list of filenames of vector files

The program mhstL calculates the Hamming distance between histograms and outputs the array of measurements to a file. Its input arguments are analogous to those of the program for vector matching (mvecL).

4.1 Program Use [mhstL]

Analogously to mvecL, we specify a histogram file (.hst) and a text file containing the file paths for a list of histogram files to be matched with:

```
> mhstL Desc/img1.hst Regist/FinasImg.txt
```

This will write the metric measurements into a file named HstLst.txt into directory Mes. You can specify a different file name by giving a third argument, ie.:

```
> mhstL Desc/img1.hst Regist/FinasImg.txt Mes/MtcImg.txt
```

The histogram files need to originate from images of (almost) similar size, in order to properly match the spatial histograms.

By default, the program matches all descriptor types, ie. contours, arc segments, straigher segments, etc. as that typically yields best results. Matlab script exsbMhstL.m (in the main folder), gives an example of how to run the process.

The program can also take focus histograms (.hsf) as input:

```
> mhstL Desc/img1_f1.hsf FinasFoc.txt
```

in which case the list of filenames in `FinasFoc.txt` need to be `hsf` files as well. For the focus histograms their original size is irrelevant, as no spatial histograms are involved; the idea of focus histograms is to create individually sized spatial histograms. Thus, the user must determine, whether it makes sense to compare two focii of different size.

4.1.1 Options

To be included.

4.2 Output

4.2.1 Program mhstL

The measurement values are written twice to file, once ordered and once unordered, both in text format. The ordered output, written to `Mes/Hst.txt`, contains two columns, one containing the index of order in zero-indexing format, and the other the distance values. The ordered file, named `Mes/HstUor.txt`, contains the measurement values in unsorted order without any other information.

Chapter 5

Focus Selection [/FocSel]

The goal of focus selection and matching is to accelerate visual browsing by maximally exploiting the descriptor output, see again the recognition pipelines suggested in Section 1.8. Structure appearing outside the image center can always be more precisely described and recognized, if we perform a saccade (change of camera direction) and then launch the descriptor extraction process again; or if we use the shape-extraction program `shpx` (Chapter 6). But that also includes more computation without having made full use of the feature description (from `dscx`). To fully exploit the richness of the descriptor output, the process of focus selection subselects the descriptors of a specified region, the *focus*, and saves them to a file for further analysis, ie. to be matched with `mvec` or `mhst`.

The program takes as input a vector file (`vec`) and a region specified as bounding box. The program then identifies the descriptors contained in that bounding box, and saves them to the file, the so-called focus file. The selection process comes in three variants:

- `focvec1` : extracts the subset of descriptor vectors for one focus, and saves them to a file with extension `vef`. That file can be loaded by `mvec` for full vector-by-vector matching.
- `fochst1` : generates an attribute histogram for one focus, and saves it to a file with extension `hsf`, useful for rapid classification.
- `fochstL` : generates attribute histograms for a list of focii as specified in a text file.

The histogram output of the latter two programs allow for rapid classification and therefore rapid visual orienting. The Matlab scripts `exsbFocVec1.m`, `exsbFocHst1.m`, `exsbFocHstL.m` and `exsbFocVecFew.m` demonstrate how to deploy those programs. They call the corresponding wrapper functions named `RennFocVec1.m`, `RennFocHst1.m` and `RennFocHstL.m`

The folder contains the following directories:

`/Desc`	vector files (as generated by `dscx`)
`/Focii`	output directory for focal selections, the `.vef` files
`/UtilMb`	Matlab scripts to read the output data files

The process extracts the corresponding part of the original feature pyramid. To illustrate that, we refer to the scheme in Section 1.3.2. For example selecting from a

quarter region of the image (with four levels), the corresponding subset of the list is organized as follows,

```
lev 2        ||
lev 1       ||||
lev 0     ||||||||
```

and starts with level 0. It reaches only level 2, as we extract a subset of the pyramid. The extracted focus pyramid has therefore three levels, nLevFoc=3.

5.1 Program Use [focvec1]

Firstly the use of focvec1 is explained. It takes three arguments:

1) a vector file as generated by dscx. The file name *must* include extension vec.
2) bounding box parameters specified as `top bottom left right`.
3) [optional] an output file name *without* extension, where the selected vectors are written to.

Example: We extract a 40x40 region (height x width) from the upper left:

```
> focvec1 Desc/img1.vec 10 50 10 50 Focii/foc1
```

The selected descriptors are then saved to directory /Focii appending the extension vef. If the output file name is not specified, the program will write to the file called Focus.vef in the same directory. How the output file is loaded will be explained in the next section.

The program calculates the number of pyramid levels automatically for the selected region. If no descriptors are found in that subspace, then no output file will be saved. More details on its output will follow below.

The program fochst1 generates a single histogram, which is written to a file with extension hsf1. The program requires the bin file with extension veb to be present (in the same directory as the vec file), as generated by dscx.

The program fochstL does the same as fochst1 but for a list of focii specified in a text file, named BboxFocii.txt, ie.

```
> fochstL Desc/img1.vec BboxFocii.txt FOCII1
```

in which the bounding boxes are given rowwise. The output is written to file FOCII1 in the example.

5.2 Output

The program writes both standard output and file output.

The standard output contains the number of levels, that was calculated automatically, nLevFoc, and the total number of descriptors detected in the focus, ntDsc, for example:

```
nLevFoc 3
ntDsc 27
```

If no descriptors are found, the output returns `ntDsc 0` and no data are written to file.

The vector values are written to the `vef` file. As mentioned already, if no output name is specified, the values will be written to a file named `Focus.vef` in the same directory. The `vef` is similar to the `vec` file as generated by `dscx`. It is loaded as demonstrated in script `LoadFocVect`, located in directory /`UtilMb`.

The example script `exsbFoc1.m` plots both the vectors of the entire image, and those of the focus (selection). The plotting routines (`PlotCntSpc`, `PlotRsgSpc`, ...) are found in directory /`Plot` of the folder for descriptor extraction (/`MtchVec`).

Chapter 6

Shape Extraction [/ShpExtr]

The goal of shape extraction is to obtain a more accurate description for a certain shape silhouette, than it was obtained with dscx. To achieve this, we specify the RGB triplet of a region (silhouette) under investigation, for example taken from the shape descriptor shp of the vector file. That region can represent anything of interest, a simple shape itself, an object silhouette, an object part, or scene part.

Using the specified RGB triplet, program shpx then carries out a simple color-segregation process, whose output can be studied with the demo program sgrRGB, to be introduced in Chapter 8. This process achieves a finer segmentation than that was obtained with dscx. Program shpx then describes the shape as was done in dscx, namely by descriptors rsg, arc and str, whose attribute values will be more accurate due to the finer outline. In addition to those descriptors, shpx also returns the spectra of the local-global space. The entire description is then saved to a file with extension shp, that can be used with the shape-matching program mshp1 (next chapter).

6.1 Program Use [shpx]

The input to the program is an image or an image patch. Taking an entire image may make sense, when the shape is large and covers almost the entire image width or height. If the shape under investigation is smaller, than it is of advantage to crop the image to its expected size, ie. extracting the individual letters of some text (in the wild). Cropping helps eliminating any distracting region. Cropping should not be too tight, as that makes proper curve description difficult; a border of at least 1 pixel should be included, even if we assume know the exact, assumed shape outline. By specifying an RGB triplet, we ensure that we deal with only that one shape. Thus, we specify for example:

```
> shpx  Imgs/ptch1.jpg 150 100 185  Desc/ptch1
```

where Imgs/ptch1.jpg is the image patch; 150, 100, 185 is the color triplet; and Desc/ptch1 is the output file stem. This will write the shape file ptch1.shp to directory /Desc. The example script exsbShpx.m runs two patches of letter shapes. If a parameter file is provided, it must follow the output file, before any long options are specified. The script rather ensures proper operation of the program; a more illustrative example is given with the demo program sgrRGB.

Depending on the complexity of the image patch, it may well be that the segmentation output identifies other, smaller regions of similar color in the image patch than the one aimed at. The matching program will take the largest one for matching.

6.1.1 Parameters and Options

Setting parameters and options is analogous to program `dscx`. An example of how to set parameter values is given with the file `Params/PrmShpx_Example.txt`. The parameter filename must contain the string `PrmShpx`. Flag `--prms` displays the parameter values used (default OFF); flag `--prmchg` displays the parameter values read by the parameter file (if provided).

6.2 Output

The shape file with extension `shp` is the only file output. For other types of output, one can deploy program `sgrRGB` (Chapter 8).

Chapter 7

Shape Matching [/ShpMtch]

The shape matching program `mshp` correlates shape descriptions as generated with program `shpx` (as introduced in the previous chapter). For a pair of shapes, three groups of measures are calculated.

1) descriptor distances: arc, straighter and radial-shape. The metric includes the angle and position attribute by default, but can be turned off by providing a parameter file containing prefered weight values.
2) spectral differences: one for the bowness spectrum and one for the straighter spectrum. This information is independent of orientation and position.
3) ratio of sizes: perimeter, height of bounding box, width of bounding box.

The measures will be outputted separately and the user can combine them to an ensemble as desired.

For the moment there exists only the one-on-one version, `mshp1`, whose usage is analogous to the vector-matching program `mvec1`.

7.1 Program Use [mshp]

Analogous to the vector matching program `mvec1`, we specify two files, in this case shape files with extension `shp`:

```
> mshp1 Desc/A.shp Desc/B.shp
```

A parameter file can be specified, that must contain the string `PrmMshp`. The example script `exsbMshp1.m` gives a simple demonstration of how to provide the parameter file and how to read the measures from the standard output.

7.2 Output

There exists only standard output. It displays the three groups of measures in three separate lines, ie.

```
mes 1.681 4.189 0.008
rts 0.847 1.033 0.969
spk 0.104 0.104
```

In detail, the values are as follows:

- `mes`: the three values arc distance, straighter distance, radial-shape distance. They can be combined to an ensemble as desired. Multiplication is a safe and uncomplicated way to obtain a reasonable prediction accuracy. A weighted sum may perform better, if the appropriate weights can be found.
- `rts`: the three values correspond to the ratios of the perimeter, height and width.
- `spk`: the two values correspond to difference of the bowness and straighter spectrum.

Chapter 8

Demo Segregation RGB
[/DemoSgrRGB]

The program sgrRGB segragates a color image into two groups according to a specified RGB triplet. Its purpose is to obtain a more precise outline of an object or scene part, than obtained by the region segmentation process (in dscx) that occurred in gray only. For example, we have detected a specific shape in the descriptor output, ie. in rsg, shp, ttg, etc. Since we know the approximate color of that shape, we can use it to cue the color-segregation process, and we obtain a more precise shape outline. The target color is assumed to represent the foreground; then, the segregation process automatically calculates a distractor target, assumed to be background. The process then performs a 2-Means clustering with a single pass (iteration). The segmented shapes of the foreground regions are then partitioned and described as it is done in program dscx, but only for the original resolution (no pyramid is generated, no divisive segmentation takes place).

To improve and accelerate the segregation process, the image should be cropped to the approximate size of the target object (or scene part). That is, the process is rather applied to only part of an image, called *patch* hereafter. The program will determine a distractor triplet using the image border pixels and for that there are different initializations possible.

The example script exsbSgrPtch1.m demonstrates how to run the program and how to load the output files. The script exsbSgrPtchInit.m compares the output for the different initialization techniques.

8.1 Program Use [sgrRGB]

The program sgrRGB takes a (color) patch as input and a RGB triplet, ie. {150, 100, 185}:

```
> sgrRGB Imgs/ptchX.jpg 150 100 185
```

This will generate the following files (there is no option for specifying a filename yet):

`BonFore.bonPix`	boundary pixels for the target/foreground
`CrvPrt.cvp`	curvature partitions (arcs and straigther segments)
`Mlab.mpu`	black-white map, with white the foreground
`Ifore.png`	the foreground regions in average color
`BonBack.bonPix`	boundary pixels for the distractor/background

How those are loaded will be explained in Section 8.2. In the following it is explained what options are available.

8.1.1 Options

The following long option is available, to be specified with a double dash '--'; single letter option are not in use.

`--init`: the initialization technique that calculates the distractor triplet, specified as digit 0, 1 or 2:

- **-0** (default): uses the (image) patch borders to calculate the average RGB value. The complexity corresponds to the number of border pixels: $O($`nPxBorder`$)$.
- **-1**: determines the farthest distance between target and each border pixel. The complexity is the same as for `init=0`: $O($`nPxBorder`$)$.
- **-2**: measures the distance of the target triplet to black {0,0,0} and white {255,255,255}, and takes the farther of either as the distractor. This has lowest complexity as it calculates only two distances: $O(2)$.

For small patches with homogeneous regions, such as a character shape in text, there is little difference between the different types of initializations. The larger the patch and the more heterogeneous the colors, the greater the differences. Since the example script runs the program on an entire image, one can observe substantial differences between different initializations.

8.2 Output

The `.cvp` file is loaded by routine `LoadCrvPrt.m` in `/DescExtr/UtilMb/Curve`.

The `.mpu` file holds the size of the map in the first two integer values, height and width. The remaining values are of type `unsigned char` and describe the map values. It is loaded with Matlab script `LoadMapUch.m` (in `/SEHBAU/UtilMb/FileIO/`). The `BonPix` files can be loaded as explained previously.

Chapter 9

Applications

We have already given an example of a generic processing pipeline for scene recognition, and one for text recognition in the wild (Section 1.8). Here we mention further applications, which will also clarify the strengths and weaknesses of the system implemented so far. We firstly discuss two prominent tasks, namely categorization and identification (Section 9.1). Then we discuss the use of structural description for navigation (Section 9.2). Finally, we highlight some other applications (Section 9.3), as well as possibilites of a fusion with other methodologies (Section 9.4).

9.1 Classification and Identification

For the classification or identification of an arbitrary scene, the pipeline as mentioned in the introduction (Section 1.8), is the most straightforward way to implement either process. Firstly we classify using the `hst` file. As pointed out already, the combination of PCA and LDA provides often good results for abstract categories, often in the range of Deep Networks that were optimized for ressources (and typically have lower recognition accuracies than full Deep Networks). Since many of our attributes describe structural biases of almost binary character, in particular the shape attributes, it is worthwhile trying classifiers that perform for such data better, such as Random Forests or Deep Belief Networks, in particular the Restricted Boltzman Machine (RBM). But also the careful development of an ensemble classifier has quite some potential to improve the classification accuracy.

With the obtained classification posteriors, one can subselect candidates for matching vectors (using `mvec`). Although the vector-matching process is better suited for identification, one can try to improve the classification result by a Nearest-Neighbor analysis.

For subtly different categories, such as the Indoor-67 collection, it requires the development of representations, that exploit the actual vector space, something that requires careful development, ie. using `mvec1`. A pragmatic choice would be to combine the result with a Convolutional Neural Network (CNN), that effortlessly learns subordinate categories (albeit at high cost and lack of structural interpretation). For that to be successful one would have to train the CNNs according to the confusions of the classification results.

For the identification of a scene, matching vectors is the immediate choice (mvec) and can be improved by matching individual regions (focii, shapes). This will work well in indoor scenes of a limited environment, for example a house or the floor of a building.

For the identification of scenes from any environment, a quasi-world recognition, the methodology of Local Features is still better. It is worthwhile combining the two approaches (Section 9.4.1). For example, one could take histogram of gradients at specific locations in the image, such as the regions provided in the saliency file (.slc), or of the contour skeleton (ACNT).

9.2 Recognition for Navigation

For an efficient navigation through an environment, it is beneficial to detect vertical and converging structures. Contour information offers a fast access to distant and thin structures, region information often holds converging lines better than contours.

Vertical structure In road scenes, there exist often thin structures at distance, ie. lamp posts, traffic sign post, poles, etc. Those appear as ridge, river or edge contours, whereby ridge and river contours conveniently delineate a structure's axis. A fast way to obtain hypotheses is to deploy the contour skeleton ACNT of the vector file and integrate across the pyramid to find strong hypotheses. Since the skeleton is already a reduced contour set, there exists the risk of misses. A more thorough way would be to use the full RRE set, which can be loaded from the vecRRE file.

In indoor scenes, structures are nearer and have larger width, ie. walls, door frames, furniture, etc. In that case, we firstly focus on region boundaries and their descriptions, such as radial descriptors ARSG, shape descriptors ASHP and tetragon descriptors ATTG. In particular the latter gives us an immediate understanding of the geometry of our surround; many of the attributes were designed for that, many more could be developed.

Converging structure Converging lines often exist as boundary segments of regions, that represent scene parts. In indoor scenes, those segments are often part of the shapes as described in list ASHP. As explained, ASHP contains by default only the inside shapes, ie. those not touching any image borders. When no inside shapes are present, as is the case in a wide-open road scene, then we include those touching the border by increasing the parameter value bordTouches. Should this still not provide candidate segments, we then analyse the full set of straighter segments in ASTRfll (of the vector file).

Road Surface The region segmentation process easily returns the regions corresponding to the pavement. The segmentation process is so sensitive, that it will also return any pavement patch with slightly different color, ie. originating from road repairs or from frequent traffic such as the rills from truck wheels. The latter is often described by ridge or river contours as well.

The challenge with this sensitive output is to distinguish between task-relevant and task-irrelevant regions. For an analysis of road surface with respect to potential damage, any region of the surface might be relevant; for driving, we want to discriminate between non-obstructive patches and those, that pose a potential danger such as a slippery surface, spilled oil, etc. Since the descriptors are generated for higher

contrasts by default, it is likely that some low-contrast regions are not described and therefore not available as radial shape rsg or abstracted shape shp. And since those descriptors probably are not sufficient for an exhaustive characterization, it is best to analyze the whole set of boundaries, available in file BonPix (Section 2.2.6).

If the scene is a dirt road, whose tracks (from previous vehicles) appear nearly equal to its surround, and therefore lacking border candidates from regions, then it is useful to focus on ridge and river contours, ie. using the full set in the vecRRE file. In this case, preprocessing the image might be of benefit, such as smoothing (low-pass filtering) the image; this facilitates finding contiguous ridge and river contours, albeit at the risk of obtaining contours that connect too much structure - as is the case for edge detection for different scales.

9.3 Other

9.3.1 Small Object Recognition

Small objects in isolation are easily detected by searching for clusters of ridge, river and edge contours. Such clusters are provided in structure YZ.Blb loaded by the saliency file (Section 2.2.3), namely the bounding boxes for attribute Num (Typ=1), as well as high-contrast clusters (Typ=2). The example script exsbPlotSalc.m demonstrates how to identify those two clusters:

```
Bnum    = YZ.Blb.Typ==1;        % numerous
Benk    = YZ.Blb.Typ==8;        % high-contrast
```

which can be used to access the bounding boxes in YZ.Blb.Box, ie.

```
YZ.Blb.Box( Bnum, : )
```

This will easily point to flying objects in the sky, floating objects in still water, etc.

The less isolated an object occurs in a scene, such as a drone flying near a tree silhouette, the more we need to know about its characteristics to discriminate it from its surround. We then build a classifier that operates on contour information, as a first step, that will helps to eliminate unlikely candidates. We increase the specificity of the classifier by including also information from radial descriptors and perhaps even shape descriptors (shp), from focii, from shape extraction (shpx), or applying the entire descriptor extraction process to solely that image part.

If an object is heavily occluded - quasi camouflaged -, such as a traffic sign covered by leaves, then we need a Deep (Network) Detector, that excels at integrating, dispersed local information. Applying a Deep Detector to the entire image is of course expensive, and also struggles with low-contrast situations, but the methodology introduced so far enables to apply the detector more specifically and therefore to accelerate its use.

9.3.2 Anomaly and Change Detection

Anomaly Detection The segmentation process is predestined for anomaly detection, because it segments anything, irrespective of contrast and shape. A transparent, plastic bag floating across the road pavement is easily detected, as well as different pavement colors as pointed out already above (Road Surface). The curve partitions (arcs and straighters) also allow to understand the structure of the anomaly. If the location of the anomaly can be determined, it makes sense to apply the segregation process sgrRGB (Section 8) or to apply shape extraction (shpx).

Change Detection The entire feature extraction output is predestined for change detection due to its thorough topological analysis. The matching program `mvec1` can be deployed for finding potential changes (Section 3.1), ie applied to focii; or the shape extraction and matching process (`shpx` and `mshp`).

9.4 Methodological Fusion

In certain scenarios, other methodologies outperform the present methodology - with its current parameter values. It therefore makes sense to combine them, as was suggested already previously. Since structure is well expressed and identified with the present methodology, the other methods can be deployed in a much more specific manner. Here we summarize potential approaches.

9.4.1 Local Features

Local features, such as histograms of gradient values, excel at identifying scenes in very large environments, ie. place recognition in the world. Those features are often sampled randomly from an image. With the contour and region features provided here, one can test a more directed approach to apply them, ie. taken at proposals as provided by the saliency file (`slc`).

9.4.2 Deep Networks/Learning

We firstly discuss the use of convolutional neural networks (CNN), that typically take pixels as input, also termed end-to-end learned. Then we mention the use of those networks that take processed input. Finally, there are possibilites to deploy Graph CNNs.

Pixels as Input (CNNs) CNNs are particularly good in two situations: for discriminating complex (multi-region) objects, or subtly different categories; and for the detection of heavily occluded objects, such as a traffic sign covered by tree branches, a situation also termed "camouflaged". Since CNNs require much ressources, it makes sense to provide candidates with our structural approach in order to accelerate the recognition process, as discussed already with the pipelines (Section 1.8). This means that one would train the CNNs according to the confusions of our structural approach. For the case of object detection, one would provide candidate locations, where the object detector is applied specifically, in order to reduce its deployment.

Processed Input For networks that take vectors as input, such as transformers, there exists of course many possibilites to test our multi-dimensional descriptor spaces. For networks that take tabular information or words as input, there exists equally many possibilities to generate them.

Graph CNNs Graph CNNs are useful for discriminating distributions in low-dimensional space. They could be applied to boundaries in order to discriminate subtly different shape silhouettes.

Appendix A

Notation

Our code notation is leaned toward the Java notation that uses concatenated, capitalized words and syllables. The underscore sign '_' we use only for function names, where a prefix denotes the type (or class) of a function or routine. For example a prefix made of a single letter such as f_, i_, u_ stands for computation, initialization and utility, respectively. But also prefixes of multiple letters are used. Only few routines are named without any underscore, that then contain a verb (or syllable of a verb) to express 'action' (ie. `Load`, `Save`, `Read`, ...) in order to distinguish themselves from variables.

- `f_Func`: routines starting with `f_` compute important functionality, such as feature extraction, feature manipulation, etc. The function name is composed of 'syllables' of three to four letters, aligned from abstract to more detailed.
- `i_Func`: routines starting with `i_` initialize an algorithm, process, etc..
- `u_Func`: functions starting with `u_` are utility functions carrying out administration, support, etc.
- `p_Func`: functions starting with `p_` are plotting routines.
- `LoadX`: loads from file with path being specified as function argument. Such functions typically call Read routines, as explained below.
- `SaveX`: saves data to file with path being specified as function argument. Such functions typically call Write routines, as explained below.
- `ReadX`: reads from file with filepointer given as function argument. They are usually called from a loading script `LoadX` (see above).
- `WriteX`: writes to file with filepointer given as function argument. They are usually called from a saving script `SaveX` (see above).
- `PlotX`: comprises a longer list of plotting instructions, calling usually plotting routines `p_Func`.
- `RennX`: runs a binary/executable from Matlab using the Matlab function `dos`. It is a wrapper routine facilitating the use of the binary with its options, e.g. `RennDscx.m` runs program `dscx`.
- `exsbX`: is a demonstration script showing how to apply a program, e.g. `exsbDscx1.m` runs program `dscx`. These scripts usually call the corresponding wrapper routine `RennX` as an example.

The following notation is used for variables:

`nX`	number of X, e.g. `nLev`
`stcX`	structure of X
`szX`	size of X, e.g. `szV`, `szH`, vertical, horizontal size

For more explanations on the notation see also my Computer Vision overview:
https://www.researchgate.net/publication/336460083