

Everything You Ever Wanted to Know About My Feature Code

I Hope.

Andrew Stein
June 2005

FEATURE EXTRACTION

Main File: `get_features.m`

Given an image, finds scale-space interest points and describes them. Interest points are computed as local maxima in a Difference-of-Gaussian pyramid. If boundary information is provided (see below), the Gaussian pyramid is computed using iterative local heat diffusion to avoid smoothing across boundaries. For each interest point found, the function determines a dominant orientation and builds a feature descriptor (local histograms of gradient orientation, weighted by gradient magnitude). This information is returned as an array of structs, where each element is a struct that represents one feature. The fields of the struct are:

- ‘x’ and ‘y’ – the location of the center of the feature in the image
- ‘scale’ – how “large” the feature is. Note that this is *not* equivalent to the size of the patch of pixels used to generate the feature descriptor and compute dominant orientation. The patch size (referred to as ‘patch_width’ in the code) is proportional to ‘scale’ and is related to the number of samples in the descriptor. The proportionality constant is a parameter in the code.
- ‘scale_index’ – This is not terribly useful, but it refers to the level in the octave of the scale space pyramid where the interest point was initially detected.
- ‘angle’ – The orientation of the descriptor, in radians.
- ‘descriptor’ – An N-vector of UINT8 values representing histogram counts of local gradient information. Currently, N is 128 (see details below for more information), but it can be changed if the descriptor is modified.

If boundary information is known, it can be passed in as an image of “crack” data. A crack image is an image of UINT8 data which encodes at each pixel whether or not there is an occlusion boundary at one (or more) of the adjacent cracks. Each of the 8 possibilities is represented by a single bit. If the bit is set, that crack is an occlusion boundary.

Bit	1	2	3	4	5	6	7	8
Crack	UP	DOWN	LEFT	RIGHT	UPLEFT	UPRIGHT	DOWNLEFT	DOWNRIGHT

So if, for example, a pixel has occlusion boundaries above and to the right (and therefore in the above-right corner as well), it will have bits 1, 3, and 6 set. So it will have a value of 37 in the crack image. **Note:** that you can convert a binary mask (or “label”) image to a crack image using the function `mask2cracks`.

Simple example calls:

```
features = get_features(img)
      or
features_boundarypreserving = get_features(img,img_cracks)
```

Sub-functions used by `get_features.m` (not necessarily exhaustive):

- **ScaleSpaceRepresentation** – builds scale space pyramids from an image (using Gaussian smoothing, or heat diffusion if boundary information is provided)

- **mexDiffuse_cracks** – heat diffusion using crack data as boundaries. Supports Neumann (replicated) or Dirichlet (zero) boundary conditions. Compile in Windows with:

```
mex -D_WIN32_ -D_NEUMANN_ diffusion_smoother_cracks.cpp -output
mexDiffuse_cracks
```

Compile in Linux with:

```
mex -D_NEUMANN_ diffusion_smoother_cracks.cpp -output
mexDiffuse_cracks
```

- **myGaussian**
- **downsample_cracks** – Downsamples a crack image by a factor of two, while preserving all of the edge structure. Used when downsampling between octaves of scale space during pyramid construction.
- **mexLocalMax** – like using Matlab’s *imregionalmax* command on a 3D cube of data, but faster. One speedup is that it ignores maxima along the borders of the data cube. Compile with:


```
mex localmax.cpp -output mexLocalMax
```
- **refine_int_pts** – finds sub-voxel interest point locations given initial [on-voxel] interest point detections from mexLocalMax
- **eliminate_edge_responses** – uses local Laplacian values to filter out interest points that are merely responses at edges
- **CreateDescriptors** – Goes through each of the interest points, extracts a patch of gradient data around it and computes the dominant orientation and feature descriptor for that patch.
 - **mexFastMarchMask_cracks** – in the case that there is crack information near an interest point, this function is called to create a boundary-respecting weight mask instead of a simple Gaussian weight mask for use in computing the descriptor. Pixels on the opposite side of the boundary from the interest point are given a lower (possibly zero) weight.
 - **CreateGaussianMask** – creates a simple Gaussian weighting mask (called when there is no nearby boundary information for the current feature)
 - **mexWeighted_hist** – used to build a weighted histogram of gradient orientations to decide a patch’s dominant orientation. When compiled with the `_INTERP_` switch, does simple linear interpolation when placing values into bins. Compile with:


```
mex -D_INTERP_ weighted_hist.cpp -output mexWeighted_hist
```
 - **smoothgradient** – Similar to Matlab’s *gradient* command, but computes gradients with Gaussian-smoothed derivatives instead of simple neighbor differences.
 - **mexInterp2** – Similar to Matlab’s *interp2* command, but much faster and more powerful for my needs. It also can return a mask indicating which requested interpolation points were out of bounds.
 - **mexComputeDescriptor_simple** – Takes a patch of gradient magnitude and orientation data, already aligned to the dominant orientation of an interest point, and computes local histograms to build a descriptor. The `#define`’s at the top can be used to specify the size of the patch being passed in, the number of histograms, and the number of orientation bins per histograms. The first time this function is run, it precomputes several static lookup values (e.g. the origins of the histograms within the patch) which are used in subsequent calls to make things faster. For a descriptor like Lowe’s, use 4x4 histograms with 8 bins each and a patch size of 17 (Lowe actually uses 16 samples, but I always use an odd patch size for various implementation reasons.)