FAST-ER Reference Manual

1.0

Generated by Doxygen 1.5.3

Wed Nov 5 10:25:01 2008

Contents

Chapter 1

FAST-ER Main Page

1.1 Copyright and License

1.2 Introduction

This project contains a suite of programs to generate an optimized corner detector, test corner detectors on a variety of datasets, generate MATLAB and optimized C++ code, and some utilities for handling the datasets. The project also includes ready to use pre-generated trees in the `fast_trees` directory. Generated trees in intermediate, C++ and MATLAB form are availavle for FAST-8, FAST-9, FAST-10, FAST-11, FAST-12 and FAST-ER. In the case of FAST-ER, the FAST detector has been learned from the best FAST-ER tree (included as `best_faster.tree`) with features extracted from all available images in the Cambridge dataset.

To make on any unix-like environment, do:

```
./configure && make
```

There is no install option.

This will create the following executables:

- `learn_detector` This learns a detector from a repeatability dataset.
- `extract_features` This extracts features from an image sequence which can be turned in to a decision tree.
- `learn_fast_tree` This learns a FAST decision tree, from extracted data.
- Programs for generating code from the learned tree, in various language/library combinations.
  - C++ / libCVD
    - `fast_tree_to_cxx_score_bsearch`
    - `fast_tree_to_cxx_score_iterate`

  - MATLAB
    * `fast_tree_to_matlab_score_bsearch`

- `test_repeatability` Measure the repeatability of a detector.
- `warp_to_png` This converts a repeatability dataset in to a rather faster loading format.
- `image_warp` This program allows visual inspection of the quality of a dataset.
- `fast_N_features` This program generates all possible FAST-N features for consumption by `learn_fast_tree`.

1.3 Requirements

This code requires the following libraries from `http://mi.eng.cam.ac.uk/~er258/cvd`

- libCVD (compiled with TooN and LAPACK support)
- TooN
- GVars3
- tag For repeatability testing, the SUSAN detector can be used if the reference implementation is downloaded and placed in the directory. It is abailable from `http://users.fmrib.ox.ac.uk/~steve/susan/susan2l.c`

1.4 Running the system

The complete sequence of operations for FAST-ER is as follows:

1. Make the executable:
   `./configure && make`
2. Generating a new FAST-ER detector.
   An example detector (the best known detector, used in the results section of the paper) is already in `best_faster.tree`.
   (a) Set up `learn_detector.cfg`. The default parameters are good, except you will need to set up the system to point to the repeatability dataset you wish to use.
   (b) Run the corner detector learning program
   `./learn_detector > logfile`
   If you run it more than once, you will probably want to alter the random seed in the configuration file.
   (c) Extract a detector from the logfile
   `awk 'a&&!NF{exit}a;/Final tree/{a=1}' logfile > new_detector.tree`
3. Measuring the repeatability of a detector
   `./test_repeatability -detector faster2 -faster2 new_detector.tree > new_detector_repeatability.txt`
   The file `new_detector_repeatability.txt` can be plotted with almost any graph plotting program. A variety of detectors can be tested using this program. See test_repeatability for more information.

4. Generating accelerated tree based detectors.

   (a) Features can be generated (for instance for FAST-N) or extracted from images, as is necessary for FAST-ER. FAST-N features can be extracted using fast_N_features:

   ```
   ./fast_N_features -N 9 > features.txt
   ```

   Alternatively, they can be extracted from images using extract_features:

   ```
   ./extract_features IMAGE1 [IMAGE2 ...] > features.txt
   ```

   (b) A decision tree can be learned from the features using learn_fast_tree:

   ```
   learn_fast_tree < features.txt > fast-tree.txt
   ```

   (c) The decision tree needs to be turned in to source code before it can be easily used. This is performed using `fast_tree_to_cxx_score_bsearch` , `fast_tree_to_cxx_score_iterate` , or `fast_tree_to_matlab_score_bsearch` .

   The name describes the target language, and the method by which the score is computed (iteraton or binary search). For monotonic trees, the result is the same, but for the more general non-monotonic trees produced by FAST-ER, the results may be slightly different.

   These programs are used in the following way:

   ```
   fast_tree_to_cxx_score_bsearch NAME fast-tree.txt > fast_tree.cxx
   ```

   NAME specifies the name of the function. If Matlab code is generated, then it is recommended that NAME is used for an output file NAME.m.

   The result is a usable source code file. In the case of generated C++, the file is compatible with libCVD, and the output of the corner detector can be fed to libCVD's nonmax_suppression function. The generated code does not make use of SSE. To do this, you will have to specify weights to learn_fast_tree and modify `fast_tree_to_cxx_score_bsearch` to hard-wire the initial questions in the tree.

   In the case of Matlab, the generated file comes with code to perform nonmaximal suppression if desired. This code is generated in straight Matlab, so corner detection will not be especially fast.

Chapter 2

FAST-ER Module Index

2.1 FAST-ER Modules

Here is a list of all modules:

Chapter 3

FAST-ER Hierarchical Index

3.1 FAST-ER Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

FAST-ER Class Index

4.1 FAST-ER Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

FAST-ER File Index

5.1 FAST-ER File List

Here is a list of all documented files with brief descriptions:

Chapter 6

FAST-ER Module Documentation

6.1 Measuring the repeatability of a detector

6.1.1 Detailed Description

Functions to load a repeatability dataset, and compute the repeatability of a list of detected points.

6.1.2 The dataset

The dataset consists of a number of registered images. The images are stored in `frames/frame_X.pgm` where X is an integer counting from zero. The frames must all be the same size. The warps are stored in `waprs/warp_Y_Z.warp`. The file `warp_Y_Z.warp` contains one line for every pixel in image Y (pixels arranged in raster-scan order). The line is the position that the pixel warps to in image Z. If location of -1, -1 indicates that this pixel does not appear in image Z.

Functions

- vector< ImageRef > generate_disc (int radius)
- Image< bool > paint_circles (const vector< ImageRef > &corners, const vector< ImageRef > &circle, ImageRef size)
- float compute_repeatability (const vector< vector< Image< array< float, 2 > > > > &warps, const vector< vector< ImageRef > > &corners, int r, ImageRef size)
- double compute_repeatability_exact (const vector< vector< Image< array< float, 2 > > > > &warps, const vector< vector< ImageRef > > &corners, double r)
- void compute_repeatability_all (const vector< Image< byte > > &images, const vector< vector< Image< array< float, 2 > > > > &warps, const DetectN &detector, const vector< int > &cpf, double fuzz)
- void compute_repeatability_noise (const vector< Image< byte > > &images, const vector< vector< Image< array< float, 2 > > > > &warps, const DetectN &detector, int cpf, float n, double fuzz)
- void mmain (int argc, char **argv)

6.1.3 Function Documentation

6.1.3.1 vector<ImageRef> generate_disc (int *radius*)

Generate a disc of ImageRefs.

Parameters:

radius Radius of the disc

Returns:

the disc of ImageRefs

Definition at line 131 of file learn_detector.cc.

Referenced by compute_repeatability().

```
132 {
133     vector<ImageRef> ret;
134     ImageRef p;
135 
136     for(p.y = -radius; p.y <= radius; p.y++)
137         for(p.x = -radius; p.x <= radius; p.x++)
138             if((int)p.mag_squared() <= radius)
139                 ret.push_back(p);
140     return ret;
141 }
```

6.1.3.2 Image<bool> paint_circles (const vector< ImageRef > & *corners*, const vector< ImageRef > & *circle*, ImageRef *size*)

Paint shapes (a vector<ImageRef>) safely in to an image This is used to paint discs at corner locations in order to perform rapid proximity checking.

Parameters:

corners Locations to paint shapes

circle Shape to paint

size Image size to be painted in to

Returns:

Image with shapes painted in to it.

Definition at line 153 of file learn_detector.cc.

Referenced by compute_repeatability().

```
154 {
155     Image<bool> im(size, 0);
156 
157 
158     for(unsigned int i=0; i < corners.size(); i++)
159         for(unsigned int j=0; j < circle.size(); j++)
160             if(im.in_image(corners[i] + circle[j]))
161                 im[corners[i] + circle[j]] = 1;
162 
163     return im;
164 }
```

6.1.3.3 float compute_repeatability (const vector< vector< Image< array< float, 2 > > > > & *warps*, const vector< vector< ImageRef > > & *corners*, int *r*, ImageRef *size*)

Computes repeatability the quick way, by caching, but has small rounding errors.

This function paints a disc of `true` around each detected corner in to an image. If a corner warps to a pixel which has the value `true` then it is a repeat.

Parameters:

warps Every warping where warps[i][j] specifies warp from image i to image j.

corners Detected corners

r A corner must be as close as this to be considered repeated

size Size of the region for cacheing. All images must be this size.

Returns:

The repeatability.

Definition at line 176 of file learn_detector.cc.

References generate_disc(), ir_rounded(), and paint_circles().

Referenced by learn_detector().

```
177 {
178     unsigned int n = corners.size();
179 
180     vector<ImageRef> disc = generate_disc(r);
181 
182     vector<Image<bool> >  detected;
183     for(unsigned int i=0; i < n; i++)
184         detected.push_back(paint_circles(corners[i], disc, size));
185 
186     int corners_tested = 0;
187     int good_corners = 0;
188 
189     for(unsigned int i=0; i < n; i++)
190         for(unsigned int j=0; j < n; j++)
191         {
192             if(i==j)
193                 continue;
194 
195             for(unsigned int k=0; k < corners[i].size(); k++)
196             {
197                 ImageRef dest = ir_rounded(warps[i][j][corners[i][k]]);
198 
199                 if(dest.x != -1)
200                 {
201                     corners_tested++;
202                     if(detected[j][dest])
203                         good_corners++;
204                 }
205             }
206         }
207 
208     return 1.0 * good_corners / (DBL_EPSILON + corners_tested);
209 }
```

6.1.3.4 double compute_repeatability_exact (const vector< vector< Image< array< float, 2 > > > > & *warps*, const vector< vector< ImageRef > > & *corners*, double *r*)

Computes repeatability the slow way to avoid rounding errors, by comparing the warped corner position to every detected corner.

A warp to x=-1, y=? is considered to be outside the image, so it is not counted.

Parameters:

warps Every warping where warps[i][j] specifies warp from image i to image j.

corners Detected corners

r A corner must be as close as this to be considered repeated

Returns:

The repeatability. No corners means zero repeatability.

Definition at line 76 of file test_repeatability.cc.

Referenced by compute_repeatability_all(), and compute_repeatability_noise().

```
77 {
78     unsigned int n = corners.size();
79
80     int repeatable_corners = 0;
81     int repeated_corners = 0;
82
83     r *= r;
84
85     for(unsigned int i=0; i < n; i++)
86         for(unsigned int j=0; j < n; j++)
87         {
88             if(i==j)
89                 continue;
90
91             for(unsigned int k=0; k < corners[i].size(); k++)
92             {
93                 const array<float, 2>& p = warps[i][j][corners[i][k]];
94
95                 if(p[0] != -1) //pixel does not warp to inside image j
96                 {
97
98                     repeatable_corners++;
99
100                     for(unsigned int l=0; l < corners[j].size(); l++)
101                     {
102                         Vector<2> d = Vec(p) - vec(corners[j][l]);
103
104                         if(d*d < r)
105                         {
106                             repeated_corners++;
107                             break;
108                         }
109                     }
110                 }
111             }
112         }
113
114     return 1.0 * (repeated_corners) / (repeatable_corners + DBL_EPSILON);
115 }
```

6.1.3.5 void compute_repeatability_all (const vector< Image< byte > > & *images*, const vector< vector< Image< array< float, 2 > > > > & *warps*, const DetectN & *detector*, const vector< int > & *cpf*, double *fuzz*)

This wrapper function computed the repeatability for a given detector and a given container of corner densities.

The result is printed to stdout.

Parameters:

images Images to test repeatability on

warps Every warping where warps[i][j] specifies warp from image i to image j.

detector Pointer to the corner detection function.

cpf The number of corners per frame to be tested.

fuzz A corner must be as close as this to be considered repeated

Definition at line 126 of file test_repeatability.cc.

References compute_repeatability_exact().

Referenced by mmain().

```
127 {
128
129     for(unsigned int i=0; i < cpf.size(); i++)
130     {
131         //Detect corners in each if the frames
132         vector<vector<ImageRef> > corners;
133         double  num_corners = 0;
134
135         for(unsigned int j=0; j < images.size(); j++)
136         {
137             vector<ImageRef> c;
138             detector(images[j], c, cpf[i]);
139             corners.push_back(c);
140
141             num_corners += c.size();
142         }
143
144         //Compute and print the repeatability.
145         cout << print << num_corners / images.size() << compute_repeatability_exact(warps, corners, fu
146     }
147 }
```

6.1.3.6 void compute_repeatability_noise (const vector< Image< byte > > & *images*, const vector< vector< Image< array< float, 2 > > > > & *warps*, const DetectN & *detector*, int *cpf*, float *n*, double *fuzz*)

This wrapper function computed the repeatability for a given detector and a given container of corner densities for variable levels of noise, from 0 to n in steps of 1 The result is printed to stdout.

Parameters:

images Images to test repeatability on

warps Every warping where warps[i][j] specifies warp from image i to image j.

detector Pointer to the corner detection function.

cpf The number of corners per frame to be tested.

n The initial noise level

fuzz A corner must be as close as this to be considered repeated

Definition at line 161 of file test_repeatability.cc.

References compute_repeatability_exact().

Referenced by mmain().

```
162 {
163
164     for(float s=0; s <= n; s++)
165     {
166
167
168         //Detect corners in each if the frames
169         vector<vector<ImageRef> > corners;
170         double  num_corners = 0;
171
172         for(unsigned int j=0; j < images.size(); j++)
173         {
174             Image<byte> ni = images[j].copy_from_me();
175
176             //Add noise to the image
177             for(Image<byte>::iterator i=ni.begin(); i != ni.end(); i++)
178                 *i = max(0, min(255, (int)floor(*i + rand_g() * s + .5)));
179
180             vector<ImageRef> c;
181             detector(ni, c, cpf);
182             corners.push_back(c);
183
184             num_corners += c.size();
185         }
186
187         //Compute and print the repeatability.
188         cout << print << s << compute_repeatability_exact(warps, corners, fuzz) << num_corners / image
189     }
190 }
```

6.1.3.7 void mmain (int *argc*, char ** *argv*)

This is the driver function.

It reads the command line arguments and calls functions to load the data and compute the repeatability.

Parameters:

argc Number of command line argumentsd.

argv Pointer to command line arguments.

Definition at line 199 of file test_repeatability.cc.

References compute_repeatability_all(), compute_repeatability_noise(), get_detector(), and load_data().

Referenced by main().

```
200 {
201     GUI.LoadFile("test_repeatability.cfg");
202     GUI.parseArguments(argc, argv);
```

```
203
204     vector<Image<byte> > images;
205     vector<vector<Image<array<float, 2> > > > warps;
206
207
208     int n = GV3::get<int>("num", 2, 1);
209     string dir = GV3::get<string>("dir", "./", 1);
210     string format = GV3::get<string>("type", "cambridge", 1);
211     double fuzz = GV3::get<double>("r", 5, 1);
212     vector<int> cpf = GV3::get<vector<int> >("cpf", "0 10 20 30 40 50 60 70 80 90 100 150 200 250 300
213     int ncpf = GV3::get<int>("ncpf", 500, 1);
214     float nmax = GV3::get<int>("nmax", 50, 1);
215     string test = GV3::get<string>("test", "normal", 1);
216
217     auto_ptr<DetectN> detector = get_detector();
218
219     rpair(images, warps) = load_data(dir, n, format);
220
221     if(test == "noise")
222         compute_repeatability_noise(images, warps, *detector, ncpf, nmax, fuzz);
223     else
224         compute_repeatability_all(images, warps, *detector, cpf, fuzz);
225
226 }
```

6.2 Repeatability dataset

6.2.1 Detailed Description

To compute repeatability, you must know for every pixel in image *A*, where that pixel ends up in image *B*.

The datasets are stored internally as:

- Images are simply stored internally as: `vector<Image<byte> >`
- Mappings from *A* to *B* are stored as: `vector<vector<array<float, 2> > >` so `mapping[i][j][y][x]`, is where pixel (x, y) in image *i* should appear in image *j*

These datasets can be stored in disk in several formats. However, they are loaded by a single function, load_data(string, int, string) In all datasets, all images must be the same size.

6.2.2 Cambridge dataset format.

The consists of *N* images, and an arbitrary warp for each image pair. From some base directory, the files are stored as:

- frames/frame_*x*.pgm
- warps/warp_*i*_*j*.warp The warp files have the mapping positions stored in row-major format, one pixel per line, stored as a pair of real numbers in text format. Details are in load_warps_cambridge() and load_images_cambridge(). The indices, *x*, *i* and *j* count from zero.

6.2.2.1 Cambridge PNG dataset.

This stores the warp data in 16 bit per channel (with a numeric range of 0–65535), colour PNG format:

- frames/frame_*x*.pgm
- pngwarps/warp_*i*_*j*.png The destination of the *x* coordinare is stored as $x = \frac{\text{red}}{\text{MULTIPLIER}} - \text{SHIFT}$, and the *y* destination as $y = \frac{\text{green}}{\text{MULTIPLIER}} - \text{SHIFT}$. MULTIPLIER is 64.0 and SHIFT is 10.0 The blue channel stores nothing. Details are in load_warps_cambridge_png().

The executable warp_to_png.cc converts a `.warp` file to a `.png` file.

6.2.3 Oxford VGG dataset format.

The datasets consist on *N* images and *N-1* Homographies describing the warps between images the first and N^{th} image. The first homography is therefore the identity matrix.

From a base directory, the files are:

- H1to*i*p
- img*i*.ppm

where the index *i* counts from 1. More details are in load_warps_cambridge_png() and load_images_vgg().

Functions

- vector< Image< byte > > load_images_cambridge (string dir, int n, string suffix)
- vector< Image< byte > > load_images_vgg (string dir, int n)
- vector< vector< Image< array< float, 2 > > > > load_warps_cambridge_png (string dir, int num, ImageRef size)
- vector< vector< Image< array< float, 2 > > > > load_warps_cambridge (string dir, int num, ImageRef size)
- vector< vector< Image< array< float, 2 > > > > load_warps_vgg (string dir, int num, ImageRef size)
- pair< vector< Image< byte > >, vector< vector< Image< array< float, 2 > > > > > load_data (string dir, int num, string format)
- void prune_warps (vector< vector< Image< array< float, 2 > > > > &warps, ImageRef size)

6.2.4 Function Documentation

6.2.4.1 vector<Image<byte> > load_images_cambridge (string *dir*, int *n*, string *suffix*)

Load images from a "Cambridge" style dataset.

Parameters:

dir The base directory of the dataset.

n The number of images in the dataset.

suffix Image filename suffix to use.

Returns:

The loaded images.

Definition at line 49 of file load_data.cc.

Referenced by load_data().

```
50 {
51     dir += "/frames/frame_%i." + suffix;
52
53     vector<Image<byte> > ret;
54
55     for(int i=0;  i < n; i++)
56     {
57         Image<byte> im;
58         im = img_load(sPrintf(dir, i));
59         ret.push_back(im);
60     }
61
62     return ret;
63 }
```

6.2.4.2 vector<Image<byte> > load_images_vgg (string *dir*, int *n*)

Load images from an "Oxford VGG" style dataset.

Parameters:

dir The base directory of the dataset.

n The number of images in the dataset.

Returns:

The loaded images.

Definition at line 72 of file load_data.cc.

Referenced by load_data().

```
73 {
74     dir += "/img%i.ppm";
75 
76     vector<Image<byte> > ret;
77 
78     for(int i=0;  i < n; i++)
79         ret.push_back(img_load(sPrintf(dir, i+1)));
80 
81     return ret;
82 }
```

6.2.4.3 vector<vector<Image<array<float,2> > > > load_warps_cambridge_png (string *dir*, int *num*, ImageRef *size*)

Load warps from a "Cambridge" repeatability dataset, with the warps stored encoded in PNG files.

See load_warps_cambridge

Parameters:

dir The base directory of the dataset.

num The numbers of images in the dataset.

size The size of the corresponding images.

Returns:

`return_value[i][j][y][x]` is where pixel x, y in image i warps to in image j.

Definition at line 113 of file load_data.cc.

Referenced by load_data().

```
114 {
115     dir += "/pngwarps/warp_%i_%i.png";
116 
117     vector<vector<Image<array<float, 2> > > > ret(num, vector<Image<array<float, 2> > >(num));
118 
119     BasicImage<byte> tester(NULL, size);
120 
121     array<float, 2> outside((TupleHead, -1, -1));
122 
123     for(int from = 0; from < num; from ++)
124         for(int to = 0; to < num; to ++)
125             if(from != to)
126             {
127                 string fname = sPrintf(dir, from, to);
128                 Image<Rgb<unsigned short> > p = img_load(fname);
129 
130                 if(p.size() != size)
```

```
131                 {
132                     cerr << "Error: warp file " << fname << " is the wrong size!\n";
133                     exit(1);
134                 }
135 
136                 Image<array<float,2> > w(size, outside);
137 
138                 for(int y=0; y < size.y; y++)
139                     for(int x=0; x < size.x; x++)
140                     {
141                         w[y][x][0] = p[y][x].red / MULTIPLIER - SHIFT;
142                         w[y][x][1] = p[y][x].green / MULTIPLIER - SHIFT;
143                     }
144 
145 
146                 cerr << "Loaded " << fname << endl;
147 
148                 ret[from][to] = w;
149             }
150 
151     return ret;
152 }
```

6.2.4.4 vector<vector<Image<array<float,2> > > > load_warps_cambridge (string *dir*, int *num*, ImageRef *size*)

Load warps from a "Cambridge" repeatability dataset.

The dataset contains warps which round to outside the image by one pixel in the max direction.

Note that the line labelled "prune" is diasbled in the evaluation of the FAST-ER system. This causes the two systems to produce slightly different results. If this line is commented out, then FAST-ER generated detectors produce exactly the same results when loaded back in to this system.

Parameters:

dir The base directory of the dataset.

num The numbers of images in the dataset.

size The size of the corresponding images.

Returns:

`return_value[i][j][y][x]` is where pixel x, y in image i warps to in image j.

Definition at line 168 of file load_data.cc.

Referenced by load_data().

```
169 {
170     dir += "/warps/warp_%i_%i.warp";
171 
172     vector<vector<Image<array<float, 2> > > > ret(num, vector<Image<array<float, 2> > >(num));
173 
174     BasicImage<byte> tester(NULL, size);
175 
176     array<float, 2> outside((TupleHead, -1, -1));
177 
178     for(int from = 0; from < num; from ++)
179         for(int to = 0; to < num; to ++)
180             if(from != to)
```

```
181             {
182                 Image<array<float,2> > w(size, outside);
183                 int n = size.x * size.y;
184                 Image<array<float,2> >::iterator p = w.begin();
185 
186                 ifstream f;
187                 string fname = sPrintf(dir, from, to);
188                 f.open(fname.c_str());
189 
190                 if(!f.good())
191                 {
192                     cerr << "Error: " << fname << ": " << strerror(errno) << endl;
193                     exit(1);
194                 }
195 
196                 array<float, 2> v;
197 
198                 for(int i=0; i < n; ++i, ++p)
199                 {
200                     f >> v;
201                     //prune
202                     //if(v[0] >= 0 && v[1] >= 0 && v[0] <= size.x-1 && v[1] <= size.y-1)
203                         *p = v;
204                 }
205 
206                 if(!f.good())
207                 {
208                     cerr << "Error: " << fname << " went bad" << endl;
209                     exit(1);
210                 }
211 
212                 cerr << "Loaded " << fname << endl;
213 
214                 ret[from][to] = w;
215             }
216 
217     return ret;
218 }
```

6.2.4.5 vector<vector<Image<array<float, 2> > > > load_warps_vgg (string *dir*, int *num*, ImageRef *size*)

Load warps from an "Oxford VGG" repeatability dataset.

The warps are stored as homographies, so warps need to be generated.

Parameters:

dir The base directory of the dataset.

num The numbers of images in the dataset.

size The size of the corresponding images.

Returns:

`return_value[i][j][y][x]` is where pixel x, y in image i warps to in image j.

Definition at line 238 of file load_data.cc.

References Arr(), and invert().

Referenced by load_data().

```
239 {
240     dir += "/H1to%ip";
241     array<float, 2> outside((TupleHead, -1, -1));
242 
243     //Load the homographies
244     vector<Matrix<3> > H_1_to_x;
245 
246     //The first homography is always the identity.
247     {
248         Matrix<3> i;
249         Identity(i);
250         H_1_to_x.push_back(i);
251     }
252 
253     for(int i=2; i <= num; i++)
254     {
255         ifstream f;
256         string fname = sPrintf(dir, i).c_str();
257         f.open(fname.c_str());
258 
259         Matrix<3> h;
260         f >> h;
261 
262         if(!f.good())
263         {
264             cerr << "Error: " << fname << " went bad" << endl;
265             exit(1);
266         }
267 
268         H_1_to_x.push_back(h);
269     }
270 
271     vector<vector<Image<array<float, 2> > > > ret(num, vector<Image<array<float, 2> > >(num));
272 
273     //Generate the warps.
274     for(int from = 0; from < num; from ++)
275         for(int to = 0; to < num; to ++)
276             if(from != to)
277             {
278                 Matrix<3> from_to_one = invert(H_1_to_x[from]);
279                 Matrix<3> one_to_to   = H_1_to_x[to];
280                 Matrix<3> from_to_to = one_to_to * from_to_one;
281 
282                 Image<array<float,2> > w(size, outside);
283 
284                 for(int y=0; y < size.y; y++)
285                     for(int x=0; x < size.x; x++)
286                     {
287                         Vector<2> p = project(from_to_to * Vector<3>((make_Vector, x, y, 1)));
288 
289                         if(p[0] >= 0 && p[1] >= 0 && p[0] <= size.x-1 && p[1] <= size.y-1)
290                             w[y][x] = Arr(p);
291                     }
292 
293                 ret[from][to] = w;
294 
295                 cerr << "Created warp " << from << " -> " << to << endl;
296             }
297 
298     return ret;
299 }
```

6.2.4.6 pair<vector<Image<byte> >, vector<vector<Image<array<float, 2> > > > > load_data (string *dir*, int *num*, string *format*)

Load a dataset.

Parameters:

dir The base directory of the dataset.

num The number of images in the dataset.

format The type of the dataset. This should be one of 'vgg', 'cam-png' or 'cam'.

Returns:

The images and the warps.

Definition at line 316 of file load_data.cc.

References load_images_cambridge(), load_images_vgg(), load_warps_cambridge(), load_warps_-cambridge_png(), and load_warps_vgg().

Referenced by main(), mmain(), and run_learn_detector().

```
317 {
318     vector<Image<byte> > images;
319     vector<vector<Image<array<float, 2> > > > warps;
320 
321     DataFormat d;
322 
323     if(format == "vgg")
324         d = VGG;
325     else if(format == "cam-png")
326         d = CambridgePNGWarp;
327     else
328         d = Cambridge;
329 
330     switch(d)
331     {
332         case Cambridge:
333             images = load_images_cambridge(dir, num, "pgm");
334             break;
335 
336         case CambridgePNGWarp:
337             images = load_images_cambridge(dir, num, "png");
338             break;
339 
340         case VGG:
341             images = load_images_vgg(dir, num);
342     };
343 
344     //Check for sanity
345     if(images.size() == 0)
346     {
347         cerr << "No images!\n";
348         exit(1);
349     }
350 
351     for(unsigned int i=0; i < images.size(); i++)
352         if(images[i].size() != images[0].size())
353         {
354             cerr << "Images are different sizes!\n";
355             exit(1);
356         }
357 
```

```
358     switch(d)
359     {
360         case CambridgePNGWarp:
361             warps = load_warps_cambridge_png(dir, num, images[0].size());
362             break;
363 
364         case Cambridge:
365             warps = load_warps_cambridge(dir, num, images[0].size());
366             break;
367 
368         case VGG:
369             warps = load_warps_vgg(dir, num, images[0].size());
370     };
371 
372 
373     return make_pair(images, warps);
374 }
```

6.2.4.7 void prune_warps (vector< vector< Image< array< float, 2 > > > > & *warps*, ImageRef *size*)

This function prunes a dataset so that no warped point will lie outside an image.

This will save on .in_image() tests later.

Parameters:

warps The warps to prune.

size the image size to prune to.

Definition at line 384 of file load_data.cc.

References ir_rounded().

Referenced by run_learn_detector().

```
385 {
386     BasicImage<byte> test(NULL, size);
387     array<float, 2> outside = make_tuple(-1, -1);
388 
389     for(unsigned int i=0; i < warps.size(); i++)
390         for(unsigned int j=0; j < warps[i].size(); j++)
391         {
392             for(Image<array<float, 2> >::iterator  p=warps[i][j].begin(); p != warps[i][j].end(); p++)
393                 if(!test.in_image(ir_rounded(*p)))
394                     *p = outside;
395         }
396 }
```

6.3 Tree representation.

Classes

- class tree_element

  This struct represents a node of the tree, and has pointers to other structs, thereby representing a branch or the entire tree.

- struct ParseError

  A named symbol to throw in the case that tree deserialization fails with a parse error.

Functions

- vector< ImageRef > tree_detect_corners_all (const Image< byte > &im, const tree_element *detector, int threshold)
- vector< ImageRef > tree_detect_corners (const Image< byte > &im, const tree_element *detector, int threshold, Image< int > scores)
- tree_element * load_a_tree (istream &i, bool eq_branch)
- tree_element * load_a_tree (istream &i)
- vector< ImageRef > transform_offsets (const vector< ImageRef > &offsets, int angle, bool r)
- void create_offsets ()

Variables

- vector< vector< ImageRef > > offsets
- int num_offsets
- pair< ImageRef, ImageRef > offsets_bbox

6.3.1 Function Documentation

6.3.1.1 vector<ImageRef> tree_detect_corners_all (const Image< byte > & *im*, const tree_element * *detector*, int *threshold*)

Detect corners without nonmaximal suppression in an image.

This contains a large amount of configurable debugging code to verify the correctness of the detector by comparing different implementations. High speed is achieved by converting the detector in to bytecode and JIT-compiling if possible.

The function recognises the following GVars:

- `debug.verify_detections` Veryify JIT or bytecode detected corners using tree_element::detect_corner

Parameters:

im The image to detect corners in.

detector The corner detector.

threshold The detector threshold.

Definition at line 45 of file faster_tree.cc.

References tree_element::bbox(), block_bytecode::detect(), tree_element::detect_corner(), and tree_element::make_fast_detector().

```
46 {
47     ImageRef tl, br, s;
48     rpair(tl,br) = detector->bbox();
49     s = im.size();
50 
51     int ymin = 1 - tl.y, ymax = s.y - 1 - br.y;
52     int xmin = 1 - tl.x, xmax = s.x - 1 - br.x;
53 
54     ImageRef pos;
55 
56     vector<int> corners;
57 
58     block_bytecode f2 = detector->make_fast_detector(im.size().x);
59 
60     f2.detect(im, corners, threshold, xmin, xmax, ymin, ymax);
61 
62 
63     if(GV3::get<bool>("debug.verify_detections"))
64     {
65         //Detect corners using slowest, but most obvious detector, since it's most likely to
66         //be correct.
67         vector<ImageRef> t;
68         for(pos.y = ymin; pos.y < ymax; pos.y++)
69         {
70             for(pos.x = xmin; pos.x < xmax; pos.x++)
71                 if(detector->detect_corner(im, pos, threshold))
72                     t.push_back(pos);
73         }
74 
75         //Verify detected corners against this result
76         if(t.size() == corners.size())
77         {
78             for(unsigned int i=0; i < corners.size(); i++)
79                 if(im.data() + corners[i] != & im[t[i]])
80                 {
81                     cerr << "Fatal error: standard and fast detectors do not match!\n";
82                     cerr << "Same number of corners, but different positions.\n";
83                     exit(1);
84                 }
85         }
86         else
87         {
88             cerr << "Fatal error: standard and fast detectors do not match!\n";
89             cerr << "Different number of corners detected.\n";
90             cerr << corners.size() << " " << t.size() << endl;
91             exit(1);
92         }
93     }
94 
95     vector<ImageRef> ret;
96 
97     int d = im.size().x;
98     for(unsigned int i=0; i < corners.size(); i++)
99     {
100         int o = corners[i];
101         ret.push_back(ImageRef(o %d, o/d));
102     }
103 
104     return ret;
105 }
```

6.3.1.2 vector<ImageRef> tree_detect_corners (const Image< byte > & *im*, const tree_element * *detector*, int *threshold*, Image< int > *scores*)

Detect corners with nonmaximal suppression in an image.

This contains a large amount of configurable debugging code to verify the correctness of the detector by comparing different implementations. High speed is achieved by converting the detector in to bytecode and JIT-compiling if possible.

The function recognises the following GVars:

- `debug.verify_detections` Veryify JIT or bytecode detected corners using tree_element::detect_corner
- `debug.verify_scores` Veryify bytecode computed scores using tree_element::detect_corner

Parameters:

im The image to detect corners in.

detector The corner detector.

threshold The detector threshold.

scores This image will be used to store the corner scores for nonmaximal suppression and is the same size as im. It is passed as a parameter since allocation of an image of this size is a significant expense.

Definition at line 125 of file faster_tree.cc.

References tree_element::bbox(), block_bytecode::detect(), tree_element::detect_corner(), and tree_element::make_fast_detector().

Referenced by learn_detector(), and faster_learn::operator()().

```
126 {
127     ImageRef tl, br, s;
128     rpair(tl,br) = detector->bbox();
129     s = im.size();
130 
131     int ymin = 1 - tl.y, ymax = s.y - 1 - br.y;
132     int xmin = 1 - tl.x, xmax = s.x - 1 - br.x;
133 
134     ImageRef pos;
135     scores.zero();
136 
137     vector<int> corners;
138 
139     block_bytecode f2 = detector->make_fast_detector(im.size().x);
140 
141     f2.detect(im, corners, threshold, xmin, xmax, ymin, ymax);
142 
143 
144     if(GV3::get<bool>("debug.verify_detections"))
145     {
146         //Detect corners using slowest, but most obvious detector, since it's most likely to
147         //be correct.
148         vector<ImageRef> t;
149         for(pos.y = ymin; pos.y < ymax; pos.y++)
150         {
151             for(pos.x = xmin; pos.x < xmax; pos.x++)
152                 if(detector->detect_corner(im, pos, threshold))
153                     t.push_back(pos);
154         }
```

```
155
156         //Verify detected corners against this result
157         if(t.size() == corners.size())
158         {
159             for(unsigned int i=0; i < corners.size(); i++)
160                 if(im.data() + corners[i] != & im[t[i]])
161                 {
162                     cerr << "Fatal error: standard and fast detectors do not match!\n";
163                     cerr << "Same number of corners, but different positions.\n";
164                     exit(1);
165                 }
166         }
167         else
168         {
169             cerr << "Fatal error: standard and fast detectors do not match!\n";
170             cerr << "Different number of corners detected.\n";
171             cerr << corners.size() << " " << t.size() << endl;
172             exit(1);
173         }
174     }
175
176
177
178     //Compute scores
179     for(unsigned int j=0; j < corners.size(); j++)
180     {
181         int i=threshold + 1;
182         while(1)
183         {
184             int n = f2.detect(im.data() + corners[j], i);
185             if(n != 0)
186                 i += n;
187             else
188                 break;
189         }
190         scores.data()[corners[j]] = i-1;
191     }
192
193     if(GV3::get<bool>("debug.verify_scores"))
194     {
195         //Compute scores using the obvious, but slow recursive implementation.
196         //This can be used to test the no obvious FAST implementation and the
197         //non obviouser JIT implementation, if it ever exists.
198         for(unsigned int j=0; j < corners.size(); j++)
199         {
200             int i=threshold + 1;
201             ImageRef pos =  im.pos(im.data() + corners[j]);
202             while(1)
203             {
204                 int n = detector->detect_corner(im, pos, i);
205                 if(n != 0)
206                     i += n;
207                 else
208                     break;
209             }
210
211             if(scores.data()[corners[j]] != i-1)
212             {
213                 cerr << "Fatal error: standard and fast scores do not match!\n";
214                 cerr << "Different score detected at  " << pos << endl;
215                 exit(1);
216             }
217         }
218     }
219
220
221     //Perform non-max suppression the simple way
```

```
222     vector<ImageRef> nonmax;
223     int d = im.size().x;
224     for(unsigned int i=0; i < corners.size(); i++)
225     {
226         int o = corners[i];
227         int v = scores.data()[o];
228 
229         if( v > *(scores.data() + o + 1    )  &&
230             v > *(scores.data() + o - 1    )  &&
231             v > *(scores.data() + o +d + 1 )  &&
232             v > *(scores.data() + o +d     )  &&
233             v > *(scores.data() + o +d - 1 )  &&
234             v > *(scores.data() + o -d + 1 )  &&
235             v > *(scores.data() + o -d     )  &&
236             v > *(scores.data() + o -d - 1))
237         {
238             nonmax.push_back(ImageRef(o %d, o/d));
239         }
240     }
241 
242     return nonmax;
243 }
```

6.3.1.3 tree_element* load_a_tree (istream & *i*, bool *eq_branch*)

Parses a tree from an istream.

This will deserialize a tree serialized by tree_element::print(). On error, ParseError is thrown.

Parameters:

i The stream to parse

eq_branch Is it an EQ branch? Check the invariant.

Returns:

An allocated tree. Ownership is passed to the callee.

Definition at line 287 of file faster_tree.cc.

References is_corner(), and split().

Referenced by faster_learn::faster_learn(), load_a_tree(), and main().

```
288 {
289     string line;
290     getline(i, line);
291 
292     vector<string> tok = split(line);
293 
294     if(tok.size() == 0)
295         throw ParseError();
296 
297     if(tok[0] == "Is")
298     {
299         if(tok.size() != 7)
300             throw ParseError();
301 
302         bool is_corner = ato<bool>(tok[2]);
303 
304         if(eq_branch && is_corner)
305         {
```

```
306             cerr << "Warning: Fixing invariant in tree\n";
307             is_corner=0;
308         }
309
310         return new tree_element(is_corner);
311     }
312     else
313     {
314         if(tok.size() != 5)
315             throw ParseError();
316
317         int offset = ato<int>(tok[0]);
318
319         auto_ptr<tree_element> t1(load_a_tree(i, false));
320         auto_ptr<tree_element> t2(load_a_tree(i, true));
321         auto_ptr<tree_element> t3(load_a_tree(i, false));
322         auto_ptr<tree_element> ret(new tree_element(t1.release(), t2.release(), t3.release(), offset))
323
324         return ret.release();
325
326     }
327 }
```

6.3.1.4 tree_element* load_a_tree (istream & *i*)

Parses a tree from an istream.

This will deserialize a tree serialized by tree_element::print(). On error, ParseError is thrown.

Parameters:

i The stream to parse

Returns:

An allocated tree. Ownership is passed to the callee.

Definition at line 334 of file faster_tree.cc.

References load_a_tree().

```
335 {
336     return load_a_tree(i, true);
337 }
```

6.3.1.5 vector<ImageRef> transform_offsets (const vector< ImageRef > & *offsets*, int *angle*, bool *r*)

Rotate a vector<ImageRef> by a given angle, with an optional reflection.

Parameters:

offsets Offsets to rotate.

angle Angle to rotate by.

r Whether to reflect.

Returns:

The rotated offsets.

Definition at line 66 of file offsets.cc.

References ir_rounded().

Referenced by create_offsets().

```
67 {
68     double a = angle * M_PI / 2;
69
70     double R_[] = { cos(a), sin(a), -sin(a) , cos(a) };
71     double F_[] = { 1, 0, 0, r?-1:1};
72
73     Matrix<2> R(R_), F(F_);
74     Matrix<2> T = R*F;
75
76     vector<ImageRef> ret;
77
78     for(unsigned int i=0; i < offsets.size(); i++)
79     {
80         Vector<2> v = vec(offsets[i]);
81         ret.push_back(ir_rounded(T * v));
82     }
83
84     return ret;
85 }
```

6.3.1.6 void create_offsets ()

Create a list of offsets with various transformation to map the offset number (see Figure 7 in the accompanying paper) to a pixel coordinate, inclusing all combinations of rotation and reflection.

The function populates offsets, and must be called before anything uses this variable.

All possible offsets are selected in an annulus, which uses the following gvars:

- `offsets.min_radius` Minimum distance from (0,0) for offset
- `offsets.max_radius` Maximum distance from (0,0) for offset

Definition at line 156 of file offsets.cc.

Referenced by main(), and run_learn_detector().

```
157 {
158     //Pixel offsets are represented as integer indices in to an array of
159     //ImageRefs. That means that by choosing the array, the tree can be
160     //rotated and/or reflected. Here, an annulus of possible offsets is
161     //created and rotated by all multiples of 90 degrees, and then reflected.
162     //This gives a total of 8.
163     offsets.resize(8);
164     {
165         double min_r = GV3::get<double>("offsets.min_radius");
166         double max_r = GV3::get<double>("offsets.max_radius");
167
168         ImageRef max((int)ceil(max_r+1), (int)ceil(max_r+1));
169         ImageRef min = -max, p = min;
170
171         //cout << "Offsets: ";
172
173         do
174         {
175             double d = vec(p) * vec(p);
176
```

```
177             if(d >= min_r*min_r && d <= max_r * max_r)
178             {
179                 offsets[0].push_back(p);
180                 //cout << offsets[0].back() << " ";
181             }
182         }
183         while(p.next(min, max));
184 
185     //  cout << endl;
186 
187         offsets_bbox = make_pair(min, max);
188     }
189     offsets[1] = transform_offsets(offsets[0], 1, 0);
190     offsets[2] = transform_offsets(offsets[0], 2, 0);
191     offsets[3] = transform_offsets(offsets[0], 3, 0);
192     offsets[4] = transform_offsets(offsets[0], 0, 1);
193     offsets[5] = transform_offsets(offsets[0], 1, 1);
194     offsets[6] = transform_offsets(offsets[0], 2, 1);
195     offsets[7] = transform_offsets(offsets[0], 3, 1);
196     num_offsets=offsets[0].size();
197 }
```

6.3.2 Variable Documentation

6.3.2.1 vector<vector<ImageRef> > offsets

Actual x,y offset of the offset numbers in the different available orientations.

Definition at line 49 of file offsets.cc.

Referenced by create_offsets(), tree_element::detect_corner(), tree_element::detect_corner_oriented(), draw_offsets(), extract_feature(), main(), tree_element::make_fast_detector(), and tree_element::make_fast_detector_o().

6.3.2.2 int num_offsets

The number of possible offsets.

Equivalent to `offsets[x].size()`

Definition at line 52 of file offsets.cc.

Referenced by create_offsets(), extract_feature(), learn_detector(), main(), and random_tree().

6.3.2.3 pair<ImageRef, ImageRef> offsets_bbox

Bounding box for offsets in all orientations.

This is therefore a bounding box for the detector.

Definition at line 55 of file offsets.cc.

Referenced by tree_element::bbox(), create_offsets(), and main().

6.4 Compiled tree representations

Classes

- struct block_bytecode

  This struct contains a byte code compiled version of the detector.

- struct block_bytecode::fast_detector_bit

  This is a bytecode element for the bytecode-compiled detector.

Functions

- block_bytecode tree_element::make_fast_detector (int xsize) const
- void tree_element::make_fast_detector_o (std::vector< block_bytecode::fast_detector_bit > &v, int n, int xsize, int N, bool invert) const

6.4.1 Function Documentation

6.4.1.1 block_bytecode tree_element::make_fast_detector (int *xsize*) const [inline, inherited]

Compile the detector to bytecode.

The bytecode is not a tree, but a graph. This is because the detector is applied in all orientations: offsets are integers which are indices in to a list of (x,y) offsets and there are multiple lists of offsets. The tree is also applied with intensity inversion.

Parameters:

xsize The width of the image.

Returns:

The bytecode compiled detector.

Definition at line 92 of file faster_tree.h.

References tree_element::gt, invert(), tree_element::lt, tree_element::make_fast_detector_o(), and offsets.

Referenced by run_learn_detector(), tree_detect_corners(), and tree_detect_corners_all().

```
93         {
94             std::vector<block_bytecode::fast_detector_bit> f;
95 
96             for(int invert=0; invert < 2; invert++)
97                 for(unsigned int i=0; i <  offsets.size(); i++)
98                 {
99                     //Make a FAST detector at a certain orientation
100                     std::vector<block_bytecode::fast_detector_bit> tmp(1);
101                     make_fast_detector_o(tmp, 0, xsize, i, invert);
102 
103                     int endpos = f.size() + tmp.size();
104                     int startpos = f.size();
105 
106                     //Append tmp on to f, filling in the non-corners (jumps to endpos)
```

```
107                     //and correcting the intermediate jumps destinations
108                     for(unsigned int i=0 ; i < tmp.size(); i++)
109                     {
110                         f.push_back(tmp[i]);
111 
112                         if(f.back().eq == -1)
113                             f.back().eq = endpos;
114                         else if(f.back().eq > 0)
115                             f.back().eq += startpos;
116 
117                         if(f.back().gt == -1)
118                             f.back().gt = endpos;
119                         else if(f.back().gt > 0)
120                             f.back().gt += startpos;
121 
122                         if(f.back().lt == -1)
123                             f.back().lt = endpos;
124                         else if(f.back().lt > 0)
125                             f.back().lt += startpos;
126 
127                     }
128                 }
129 
130             //We need a final endpoint for non-corners
131             f.resize(f.size() + 1);
132             f.back().offset = 0;
133             f.back().lt = 0;
134             f.back().gt = 0;
135             f.back().eq = 0;
136 
137             //Now we need an extra endpoint for corners
138             for(unsigned int i=0; i < f.size(); i++)
139             {
140                 //EQ is always non-corner
141                 if(f[i].lt == -2)
142                     f[i].lt = f.size();
143                 if(f[i].gt == -2)
144                     f[i].gt = f.size();
145             }
146 
147             f.resize(f.size() + 1);
148             f.back().offset = 0;
149             f.back().lt = 0;
150             f.back().gt = 1;
151             f.back().eq = 0;
152 
153             block_bytecode r = {f};
154 
155             return r;
156         }
```

6.4.1.2 void tree_element::make_fast_detector_o (std::vector< block_bytecode::fast_detector_bit > & *v*, int *n*, int *xsize*, int *N*, bool *invert*) const [inline, private, inherited]

This compiles the tree in a single orientation and form to bytecode.

This is called repeatedly by make_fast_detector. A jump destination of -1 refers to a non corner and a destination of -2 refers to a corner.

Parameters:

- ***v*** Bytecode storage
- ***n*** Position in v to compile the bytecode to

xsize Width of the image

N orientation of the tree

invert whether or not to perform and intensity inversion.

Definition at line 175 of file faster_tree.h.

References tree_element::eq, tree_element::gt, tree_element::is_corner, tree_element::is_leaf(), tree_element::lt, tree_element::make_fast_detector_o(), tree_element::offset_index, and offsets.

Referenced by tree_element::make_fast_detector(), and tree_element::make_fast_detector_o().

```
176         {
177             //-1 for non-corner
178             //-2 for corner
179 
180             if(eq == NULL)
181             {
182                 //If the tree is a single leaf, then we end up here. In this case, it must be
183                 //a non-corner, otherwise the strength would be inf.
184                 v[n].offset = 0;
185                 v[n].lt = -1;
186                 v[n].gt = -1;
187                 v[n].eq = -1;
188             }
189             else
190             {
191                 v[n].offset = offsets[N][offset_index].x + offsets[N][offset_index].y * xsize;
192 
193                 if(eq->is_leaf())
194                     v[n].eq = -1; //Can only be non-corner!
195                 else
196                 {
197                     v[n].eq = v.size();
198                     v.resize(v.size() + 1);
199                     eq->make_fast_detector_o(v, v[n].eq, xsize, N, invert);
200                 }
201 
202                 const tree_element* llt = lt;
203                 const tree_element* lgt = gt;
204 
205                 if(invert)
206                     std::swap(llt, lgt);
207 
208                 if(llt->is_leaf())
209                 {
210                     v[n].lt = -1 - llt->is_corner;
211                 }
212                 else
213                 {
214                     v[n].lt = v.size();
215                     v.resize(v.size() + 1);
216                     llt->make_fast_detector_o(v, v[n].lt, xsize, N, invert);
217                 }
218 
219 
220                 if(lgt->is_leaf())
221                     v[n].gt = -1 - lgt->is_corner;
222                 else
223                 {
224                     v[n].gt = v.size();
225                     v.resize(v.size() + 1);
226                     lgt->make_fast_detector_o(v, v[n].gt, xsize, N, invert);
227                 }
228             }
229         }
```

6.5 Utility functions.

Defines

- #define fatal(E, S,...) vfatal((E), (S), (tag::Fmt,## __VA_ARGS__))

Functions

- vector< string > split (const string &s)
- template<class C>
  C ato (const string &s)
- double sq (double d)
- vector< int > range (int num)
- template<class C>
  void vfatal (int err, const string &s, const C &list)
- istream & operator>> (istream &i, array< float, 2 > &f)
- array< float, 2 > Arr (const Vector< 2 > &vec)
- Matrix< 3 > invert (const Matrix< 3 > &m)
- void draw_offset_list (const vector< ImageRef > &offsets)
- void draw_offsets ()
- CVD::ImageRef ir_rounded (const tag::array< float, 2 > &v)

6.5.1 Define Documentation

6.5.1.1 #define fatal(E, S, ...) vfatal((E), (S), (tag::Fmt,## __VA_ARGS__))

Print an error message and the exit.

Parameters:

E Error code

S Format string

Definition at line 125 of file learn_fast_tree.cc.

Referenced by find_best_split(), load_features(), main(), and datapoint< FEATURE_SIZE >::pack_trits().

6.5.2 Function Documentation

6.5.2.1 vector<string> split (const string & *s*)

Tokenise a string.

Parameters:

s String to be split

Returns:

Tokens

Definition at line 249 of file faster_tree.cc.

Referenced by load_a_tree().

```
250 {
251     istringstream i(s);
252
253     vector<string> v;
254
255     while(!i.eof())
256     {
257         string s;
258         i >> s;
259         if(s != "")
260             v.push_back(s);
261     }
262     return v;
263 }
```

6.5.2.2 template<class C> C ato (const string & *s*) [inline]

String to some class.

Name modelled on atoi.

Parameters:

s String to parse

Returns:

class the string was parsed in to.

Definition at line 269 of file faster_tree.cc.

```
270 {
271     istringstream i(s);
272     C c;
273     i >> c;
274
275     if(i.bad())
276         throw   ParseError();
277
278     return c;
279 }
```

6.5.2.3 double sq (double *d*)

Square a number.

Parameters:

d Number to square

Returns:

d^2

Definition at line 100 of file learn_detector.cc.

Referenced by learn_detector().

```
101 {
102     return d*d;
103 }
```

6.5.2.4 vector<int> range (int *num*)

Populate a std::vector with the numbers 0,1,.

..,num

Parameters:

num Size if the range

Returns:

the populated vector.

Definition at line 110 of file learn_detector.cc.

```
111 {
112     vector<int> r;
113
114     for(int i=0; i < num; i++)
115         r.push_back(i);
116     return r;
117 }
```

6.5.2.5 template<class C> void vfatal (int *err*, const string & *s*, const C & *list*) [inline]

Print an error message and the exit, using Tuple stype VARARGS.

Parameters:

err Error code

s Format string

list Argument list

Definition at line 131 of file learn_fast_tree.cc.

```
132 {
133     vfPrintf(cerr, s + "\n", list);
134     exit(err);
135 }
```

6.5.2.6 istream& operator>> (istream & *i*, array< float, 2 > & *f*)

Load an array from an istream.

Parameters:

i Stream to load from

f array to load in to

Definition at line 88 of file load_data.cc.

```
89 {
90     i >> f[0] >> f[1];
91     return i;
92 }
```

6.5.2.7 array<float, 2> Arr (const Vector< 2 > & *vec*)

Convert a vector in to an array.

Parameters:

vec Vector to convert

Definition at line 97 of file load_data.cc.

Referenced by load_warps_vgg().

```
98 {
99     return array<float, 2>((TupleHead, vec[0], vec[1]));
100 }
```

6.5.2.8 Matrix<3> invert (const Matrix< 3 > & *m*)

Invert a matrix.

Parameters:

m Matrix to invert

Definition at line 223 of file load_data.cc.

Referenced by tree_element::detect_corner(), load_warps_vgg(), and tree_element::make_fast_detector().

```
224 {
225     LU<3> i(m);
226     return i.get_inverse();
227 }
```

6.5.2.9 void draw_offset_list (const vector< ImageRef > & *offsets*)

Pretty print some offsets to stdout.

Parameters:

offsets List of offsets to pretty-print.

Definition at line 91 of file offsets.cc.

Referenced by draw_offsets().

```
92 {
93
94     cout << "Allowed offsets: " << offsets.size() << endl;
95
96     ImageRef min, max;
97     min.x = *min_element(member_iterator(offsets.begin(), &ImageRef::x), member_iterator(offsets.end(),
98     max.x = *max_element(member_iterator(offsets.begin(), &ImageRef::x), member_iterator(offsets.end(),
99     min.y = *min_element(member_iterator(offsets.begin(), &ImageRef::y), member_iterator(offsets.end(),
100     max.y = *max_element(member_iterator(offsets.begin(), &ImageRef::y), member_iterator(offsets.end()
101
102     cout << print << min <<max << endl;
103
104     Image<int> o(max-min+ImageRef(1,1), -1);
105     for(unsigned int i=0; i <offsets.size(); i++)
106         o[offsets[i] -min] = i;
107
108     for(int y=0; y < o.size().y; y++)
109     {
110         for(int x=0; x < o.size().x; x++)
111             cout << "+------";
112         cout << "+"<< endl;
113
114         for(int x=0; x < o.size().x; x++)
115             cout << "|      ";
116         cout << "|"<< endl;
117
118
119         for(int x=0; x < o.size().x; x++)
120         {
121             if(o[y][x] >= 0)
122                 cout << "|  " << setw(2) << o[y][x] << "  ";
123             else if(ImageRef(x, y) == o.size() / 2)
124                 cout << "|   " << "#" << "  ";
125             else
126                 cout << "|      ";
127         }
128         cout <<  "|" << endl;
129
130         for(int x=0; x < o.size().x; x++)
131             cout << "|      ";
132         cout << "|"<< endl;
133     }
134
135     for(int x=0; x < o.size().x; x++)
136         cout << "+------";
137     cout << "+"<< endl;
138
139     cout << endl;
140
141 }
```

6.5.2.10 void draw_offsets ()

Prettyprints the contents of offsets.

Definition at line 201 of file offsets.cc.

Referenced by run_learn_detector().

```
202 {
203     //Print the offsets out.
204     for(unsigned int i=0; i < 8; i++)
205     {
206         cout << "Offsets " << i << endl;
207         draw_offset_list(offsets[i]);
208         cout << endl;
209     }
210 }
```

6.5.2.11 CVD::ImageRef ir_rounded (const tag::array< float, 2 > & *v*) [inline]

Convert a float array into an image co-ordinate.

Numbers are rounded

Parameters:

v The array to convert

Definition at line 30 of file utility.h.

Referenced by compute_repeatability(), prune_warps(), and transform_offsets().

```
31 {
32     return CVD::ImageRef(
33   static_cast<int>(v[0] > 0.0 ? v[0] + 0.5 : v[0] - 0.5),
34   static_cast<int>(v[1] > 0.0 ? v[1] + 0.5 : v[1] - 0.5));
35 }
```

6.6 Functions for detecting corners of various types.

Classes

- struct SearchThreshold

  This class wraps a DetectT class with binary_search_threshold and presents is as a DetectN class.

- struct Random

  Detector which randomly scatters corners around an image.

- struct DetectN

  A corner detector object which is passed a target number of corners to detect.

- struct DetectT

  A corner detector object which is passed a threshold.

- struct dog

  Class wrapping the Difference of Gaussians detector.

- struct harrisdog

  Class wrapping the Harris-Laplace detector.

- struct faster_learn

  FAST-ER detector.

- struct ShiTomasiDetect

  Class wrapping the Harris detector.

- struct HarrisDetect

  Class wrapping the Shi-Tomasi detector.

- struct SUSAN

  Class wrapping the SUSAN detector.

Functions

- int binary_search_threshold (const Image< byte > &i, vector< ImageRef > &c, unsigned int N, const DetectT &detector)
- auto_ptr< DetectN > get_detector ()
- void HarrisDetector (const CVD::Image< float > &i, std::vector< std::pair< float, CVD::ImageRef > > &c, unsigned int N, float blur, float sigmas)

6.6.1 Function Documentation

6.6.1.1 int binary_search_threshold (const Image< byte > & *i*, vector< ImageRef > & *c*, unsigned int *N*, const DetectT & *detector*)

This takes a detector which requires a threshold and uses binary search to get as close as possible to the requested number of corners.

Parameters:

i The image in which to detect corners.

c The detected corners to be returned.

N The target number of corners.

detector The corner detector.

Definition at line 47 of file detectors.cc.

Referenced by SearchThreshold::operator()().

```
48 {
49     //Corners for high, low and midpoint thresholds.
50     vector<ImageRef> ch, cl, cm;
51 
52     //The high and low thresholds.
53     unsigned int t_high = 256;
54     unsigned int t_low = 0;
55 
56 
57     detector(i, ch, t_high);
58     detector(i, cl, t_low);
59 
60     while(t_high > t_low + 1)
61     {
62 
63         cm.clear();
64         unsigned int t = (t_high + t_low    ) / 2;
65         detector(i, cm, t);
66 
67         if(cm.size() == N)
68         {
69             c = cm;
70             return t;
71         }
72         else if(cm.size() < N) //If we detected too few points, then the t is too high
73         {
74             t_high = t;
75             ch = cm;
76         }
77         else //We detected too many points to t is too low.
78         {
79             t_low = t;
80             cl = cm;
81         }
82     }
83 
84     //Pick the closest
85     //If there is ambiguity, go with the lower threshold (more corners).
86     //The only reason for this is that the evaluation code in the FAST-ER
87     //system uses this rule.
88     if( N - ch.size() >= cl.size() - N)
89     {
90         c = cl;
91         return t_low;
92     }
93     else
94     {
95         c = ch;
96         return t_high;
97     }
98 }
```

6.6.1.2 auto_ptr<DetectN> get_detector ()

Very simple factory function for getting detector objects.

Get a corner detector.

Paramaters (including the detector type) are drawn from the GVars database. The parameter "detector" determines the detector type. Valid options are:

- random Randomly scatter corners around the image.
- dog Difference of Gaussians detector
- harrisdog Harris-Laplace (actually implemented as Harris-DoG) detector
- harris Harris detector with Gaussian blur
- shitomasi Shi-Tomasi detector
- susan Reference implementation of the SUSAN detector
- fast9 libCVD's builtin FAST-9 detector
- fast9old libCVD's builtin FAST-9 detector with the old scoring algorithm, as seen in [Rosten, Drummond 2006].
- fast12 libCVD's builtin FAST-12 detector
- faster2 A FAST-ER detector loaded from a file containing the tree

Definition at line 156 of file detectors.cc.

Referenced by mmain().

```
157 {
158
159     string d = GV3::get<string>("detector", "fast9", 1);
160
161     if(d == "random")
162         return auto_ptr<DetectN>(new Random);
163     else if(d == "dog")
164         return auto_ptr<DetectN>(new dog);
165     else if(d == "harrisdog")
166         return auto_ptr<DetectN>(new harrisdog);
167     else if(d == "shitomasi")
168         return auto_ptr<DetectN>(new ShiTomasiDetect);
169     else if(d == "harris")
170         return auto_ptr<DetectN>(new HarrisDetect);
171     #ifdef USESUSAN
172         else if(d == "susan")
173             return auto_ptr<DetectN>(new SearchThreshold(new SUSAN));
174     #endif
175     else if(d == "fast9")
176         return auto_ptr<DetectN>(new SearchThreshold(new fast_9));
177     else if(d == "fast9old")
178         return auto_ptr<DetectN>(new SearchThreshold(new fast_9_old));
179     else if(d == "fast12")
180         return auto_ptr<DetectN>(new SearchThreshold(new fast_12));
181     else if(d == "faster2")
182         return auto_ptr<DetectN>(new SearchThreshold(new faster_learn(GV3::get<string>("faster2"))));
183     else
184     {
185         cerr << "Unknown detector: " << d << endl;
186         exit(1);
187     }
188 }
```

6.6.1.3 void HarrisDetector (const CVD::Image< float > & *i*, std::vector< std::pair< float, CVD::ImageRef > > & *c*, unsigned int *N*, float *blur*, float *sigmas*)

Detect Harris corners.

Parameters:

i Image in which to detect corners

c Detected corners and strengths are inserted in to this container

N Number of corners to detect

blur Standard deviation of blur to use

sigmas Blur using sigmas standard deviations

Referenced by harrisdog::operator()().

6.7 Optimization routines

6.7.1 Detailed Description

The functions in this section deal specifically with optimizing a decision tree detector for repeatability.

The code in learn_detector() is a direct implementation of the algorithm described in section V of the accompanying paper.

The manipulation of the tree is necessarily tied to the internal representation which is described in the tree representation.

Functions

- tree_element * random_tree (int d, bool is_eq_branch=1)
- double compute_temperature (int i, int imax)
- tree_element * learn_detector (const vector< Image< byte > > &images, const vector< vector< Image< array< float, 2 > > > > &warps)
- void run_learn_detector (int argc, char **argv)
- int main (int argc, char **argv)

6.7.2 Function Documentation

6.7.2.1 tree_element* random_tree (int *d*, bool *is_eq_branch* = 1)

Generate a random tree, as part of a stochastic optimization scheme.

Parameters:

d Depth of tree to generate

is_eq_branch Whether eq-branch constraints should be applied. This should always be true when the function is called.

Definition at line 218 of file learn_detector.cc.

References num_offsets.

Referenced by learn_detector().

```
219 {
220     //Recursively generate a tree of depth d
221     //
222     //Generated trees respect invariant 1
223     if(d== 0)
224         if(is_eq_branch)
225             return new tree_element(0);
226         else
227             return new tree_element(rand()%2);
228     else
229         return new tree_element(random_tree(d-1, 0), random_tree(d-1, 1), random_tree(d-1, 0), rand()%
230 }
```

6.7.2.2 double compute_temperature (int *i*, int *imax*)

Compute the current temperature from parameters in the configuration file.

Parameters:

i The current iteration.

imax The maximum number of iterations.

Returns:

The temperature.

Definition at line 241 of file learn_detector.cc.

Referenced by learn_detector().

```
242 {
243     double scale=GV3::get<double>("Temperature.expo.scale");
244     double alpha = GV3::get<double>("Temperature.expo.alpha");
245 
246     return scale * exp(-alpha * i / imax);
247 }
```

6.7.2.3 tree_element* learn_detector (const vector< Image< byte > > & *images*, const vector< vector< Image< array< float, 2 > > > > & *warps*)

Generate an optimized corner detector.

Parameters:

images The training images

warps Warps for evaluating the performance on the training images.

Returns:

An optimized detector.

Definition at line 258 of file learn_detector.cc.

References compute_repeatability(), compute_temperature(), tree_element::copy(), tree_element::eq, tree_element::gt, tree_element::is_corner, tree_element::is_leaf(), tree_element::lt, tree_element::nth_element(), tree_element::num_nodes(), num_offsets, tree_element::offset_index, tree_element::print(), random_tree(), sq(), and tree_detect_corners().

Referenced by run_learn_detector().

```
259 {
260     unsigned int  iterations=GV3::get<unsigned int>("iterations");       // Number of iterations of si
261     int threshold = GV3::get<int>("FAST_threshold");                     // Threshold at which to perf
262     int fuzz_radius=GV3::get<int>("fuzz");                               // A point must be this close
263     double repeatability_scale = GV3::get<double>("repeatability_scale");// w_r
264     double num_cost =   GV3::get<double>("num_cost");                    // w_n
265     int max_nodes = GV3::get<int>("max_nodes");                          // w_s
266 
267     bool first_time = 1;
268     double old_cost = HUGE_VAL;                                          //This will store the final s
```

```
269
270     ImageRef image_size = images[0].size();
271
272     set<int> debug_triggers = GV3::get<set<int> >("triggers");           //Allow artitrary GVars code
273
274     //Preallocated space for nonmax-suppression. See tree_detect_corners()
275     Image<int> scratch_scores(image_size, 0);
276
277     //Start with an initial random tree
278     tree_element* tree = random_tree(GV3::get<int>("initial_tree_depth"));
279
280     for(unsigned int itnum=0; itnum < iterations; itnum++)
281     {
282         if(debug_triggers.count(itnum))
283             GUI.ParseLine(GV3::get<string>(sPrintf("trigger.%i", itnum)));
284
285         /* Trees:
286
287             Invariants:
288                 1:     eq->{0,0,0,(0,0),0}          //Leafs of an eq pointer must not be corners
289
290             Operations:
291                 Leaves:
292                     1: Splat on a random subtree of depth 1 (respect invariant 1)
293                     2: Flip  class   (respect invariant 1)
294
295                 Nodes:
296                     3: Copy one subtree to another subtree (no invariants need be respected)
297                     4: Randomize offset (no invariants need be respected)
298                     5: Splat a subtree in to a single node.
299
300
301         Cost:
302
303           (1 + (#nodes/max_nodes)^2) * (1 - repeatability)^2 * Sum_{frames} exp(- (fast_9_num-detected
304
305         */
306
307
308         //Deep copy in to new_tree and work with the copy.
309         tree_element* new_tree = tree->copy();
310
311         cout << "\n\n-------------------------------------\n";
312         cout << print << "Iteration" << itnum;
313
314         if(GV3::get<bool>("debug.print_old_tree"))
315         {
316             cout << "Old tree is:" << endl;
317             tree->print(cout);
318         }
319
320         //Skip tree modification first time so that the randomly generated
321         //initial tree can be evaluated
322         if(!first_time)
323         {
324
325             //Create a tree permutation
326             tree_element* node;
327             bool node_is_eq;
328
329
330             //Select a random node
331             int nnum = rand() % new_tree->num_nodes();
332             rpair(node, node_is_eq) = new_tree->nth_element(nnum);
333
334             cout << "Permuting tree at node " << nnum << endl;
335             cout << print << "Node" << node << node_is_eq;
```

```
336
337
338             //See section 4 in the paper.
339             if(node->eq == NULL) //A leaf
340             {
341                 if(rand() % 2 || node_is_eq)  //Operation 1, invariant 1
342                 {
343                     cout << "Growing a subtree:\n";
344                     //Grow a subtree
345                     tree_element* stub = random_tree(1);
346
347                     stub->print(cout);
348
349                     //Splice it on manually (ick)
350                     *node = *stub;
351                     stub->lt = stub->eq = stub->gt = 0;
352                     delete stub;
353
354                 }
355                 else //Operation 2
356                 {
357                     cout << "Flipping the classification\n";
358                     node->is_corner  = ! node->is_corner;
359                 }
360             }
361             else //A node
362             {
363                 double d = rand_u();
364
365                 if(d < 1./3.) //Randomize the test
366                 {
367                     cout << "Randomizing the test\n";
368                     node->offset_index = rand() % num_offsets;
369                 }
370                 else if(d < 2./3.)
371                 {
372                     //Select r, c \in {0, 1, 2} without replacement
373                     int r = rand() % 3; //Remove
374                     int c;              //Copy
375                     while((c = rand()%3) == r){}
376
377                     cout << "Copying branches " << c << " to " << r <<endl;
378
379                     //Deep copy node c: it's a tree, not a graph.
380                     tree_element* tmp;
381
382                     if(c == 0)
383                         tmp = node->lt->copy();
384                     else if(c == 1)
385                         tmp = node->eq->copy();
386                     else
387                         tmp = node->gt->copy();
388
389                     //Delete r and put the copy of c in its place
390                     if(r == 0)
391                     {
392                         delete node->lt;
393                         node->lt = tmp;
394                     }
395                     else if(r == 1)
396                     {
397                         delete node->eq;
398                         node->eq = tmp;
399                     }
400                     else
401                     {
402                         delete node->gt;
```

```
403                         node->gt = tmp;
404                     }
405
406                     //NB BUG!!!
407                     //At this point the invariant can be broken,
408                     //since a "corner" leaf could have been copied
409                     //to an "eq" branch.
410
411                     //Oh dear. This bug made it in to the paper.
412                     //Fortunately, the bytecode compiler ignores the tree
413                     //when it can decuce its structure from the invariant.
414
415                     //The following line should have been present in the paper:
416                     if(node->eq->is_leaf())
417                         node->eq->is_corner = 0;
418
419                     //Happily, because the bytecode compiler deduces this
420                     //it behaves as if this line was present, at evaluation time.
421                     //Of course, the presense of this line will produce different
422                     //results later if the node is subsequently copied back in one
423                     //of these operations.
424                 }
425                 else //Splat!!! ie delete a subtree
426                 {
427                     cout << "Splat!!!1\n";
428                     delete node->lt;
429                     delete node->eq;
430                     delete node->gt;
431                     node->lt = node->eq = node->gt = 0;
432
433                     if(node_is_eq) //Maintain invariant 1
434                         node->is_corner = 0;
435                     else
436                         node->is_corner = rand()%2;
437                 }
438             }
439         }
440         first_time=0;
441
442         if(GV3::get<bool>("debug.print_new_tree"))
443         {
444             cout << "New tree is: "<< endl;
445             new_tree->print(cout);
446         }
447
448
449         //Detect all corners in all images
450         vector<vector<ImageRef> > detected_corners;
451         for(unsigned int i=0; i < images.size(); i++)
452             detected_corners.push_back(tree_detect_corners(images[i], new_tree, threshold, scratch_scc
453
454
455         //Compute repeatability and assosciated cost
456         double repeatability = compute_repeatability(warps, detected_corners, fuzz_radius, image_size)
457         double repeatability_cost = 1 + sq(repeatability_scale/repeatability);
458
459         //Compute cost associated with the total number of detected corners.
460         float number_cost=0;
461         for(unsigned int i=0; i < detected_corners.size(); i++)
462         {
463             double cost = sq(detected_corners[i].size() / num_cost);
464             cout << print << "Image" << i << detected_corners[i].size()<< cost;
465             number_cost += cost;
466         }
467         number_cost = 1 + number_cost / detected_corners.size();
468         cout << print << "Number cost" << number_cost;
469
```

```
470         //Cost associated with tree size
471         double size_cost = 1 + sq(1.0 * new_tree->num_nodes()/max_nodes);
472
473         //The overall cost function
474         double cost = size_cost * repeatability_cost * number_cost;
475
476         double temperature = compute_temperature(itnum,iterations);
477
478
479         //The Boltzmann acceptance criterion:
480         //If cost < old cost, then old_cost - cost > 0
481         //so exp(.) > 1
482         //so drand48() < exp(.) == 1
483         double liklihood=exp((old_cost-cost) / temperature);
484
485
486         cout << print << "Temperature" << temperature;
487         cout << print << "Number cost" << number_cost;
488         cout << print << "Repeatability" << repeatability << repeatability_cost;
489         cout << print << "Nodes" << new_tree->num_nodes() << size_cost;
490         cout << print << "Cost" << cost;
491         cout << print << "Old cost" << old_cost;
492         cout << print << "Liklihood" << liklihood;
493
494         //Make the Boltzmann decision
495         if(rand_u() < liklihood)
496         {
497             cout << "Keeping change" << endl;
498             old_cost = cost;
499             delete tree;
500             tree = new_tree;
501         }
502         else
503         {
504             cout << "Rejecting change" << endl;
505             delete new_tree;
506         }
507
508         cout << print << "Final cost" << old_cost;
509
510     }
511
512
513     return tree;
514 }
```

6.7.2.4 void run_learn_detector (int *argc*, char ** *argv*)

Load configuration and data and learn a detector.

Parameters:

argc Number of command line arguments

argv Vector of command line arguments

Definition at line 522 of file learn_detector.cc.

References create_offsets(), draw_offsets(), learn_detector(), load_data(), tree_element::make_fast_-detector(), block_bytecode::print(), tree_element::print(), and prune_warps().

Referenced by main().

```
523 {
```

```
524
525     //Process configuration information
526     GUI.LoadFile("learn_detector.cfg");
527     GUI.parseArguments(argc, argv);
528
529
530     //Load a ransom seed.
531     if(GV3::get<int>("random_seed") != -1)
532         srand(GV3::get<int>("random_seed"));
533
534     //Initialize the global information for the tree
535     create_offsets();
536     draw_offsets();
537
538
539     //Load the training set
540     string dir=GV3::get<string>("repeatability_dataset.directory");
541     string format=GV3::get<string>("repeatability_dataset.format");
542     int num=GV3::get<int>("repeatability_dataset.size");
543
544     vector<Image<byte> > images;
545     vector<vector<Image<array<float, 2> > > > warps;
546
547     rpair(images, warps) = load_data(dir, num, format);
548
549     prune_warps(warps, images[0].size());
550
551
552     //Learn a detector
553     tree_element* tree = learn_detector(images, warps);
554
555     //Print out the results
556     cout << "Final tree is:" << endl;
557     tree->print(cout);
558     cout << endl;
559
560     cout << "Final block detector is:" << endl;
561     {
562         block_bytecode f = tree->make_fast_detector(9999);
563         f.print(cout, 9999);
564     }
565 }
```

6.7.2.5 int main (int *argc*, char ** *argv*)

Driver wrapper.

Parameters:

argc Number of command line arguments

argv Vector of command line arguments

Definition at line 572 of file learn_detector.cc.

References run_learn_detector().

```
573 {
574     try
575     {
576         run_learn_detector(argc, argv);
577     }
578     catch(Exceptions::All w)
```

```
579     {
580         cerr << "Error: " << w.what << endl;
581     }
582 }
```

Chapter 7

FAST-ER Class Documentation

7.1 block_bytecode Struct Reference

`#include <faster_bytecode.h>`

7.1.1 Detailed Description

This struct contains a byte code compiled version of the detector.

Definition at line 35 of file faster_bytecode.h.

Public Member Functions

- bool detect_no_score (const CVD::byte *imp, int b) const
- int detect (const CVD::byte *imp, int b) const
- void print (std::ostream &o, int width) const
- void detect (const CVD::Image< CVD::byte > &im, std::vector< int > &corners, int threshold, int xmin, int xmax, int ymin, int ymax)

Public Attributes

- std::vector< fast_detector_bit > d

Classes

- struct fast_detector_bit

  This is a bytecode element for the bytecode-compiled detector.

7.1.2 Member Function Documentation

7.1.2.1 bool block_bytecode::detect_no_score (const CVD::byte * *imp*, int *b*) const [inline]

Detects a corner at a given pointer, without the book keeping required to compute the score.

This is quite a lot faster than detect.

Parameters:

imp Pointer at which to detect corner

b FAST barrier

Returns:

is a corner or not

Definition at line 72 of file faster_bytecode.h.

References d.

Referenced by detect().

```
73     {
74         int n=0;
75         int cb = *imp + b;
76         int c_b = *imp - b;
77         int p;
78 
79         while(d[n].lt)
80         {
81             p = imp[d[n].offset];
82 
83             if(p > cb)
84                 n = d[n].gt;
85             else if(p < c_b)
86                 n = d[n].lt;
87             else
88                 n = d[n].eq;
89         }
90 
91         return d[n].gt;
92     }
```

7.1.2.2 int block_bytecode::detect (const CVD::byte * *imp*, int *b*) const [inline]

Detects a corner at a given pointer, with book-keeping required for score computation.

Parameters:

imp Pointer at which to detect corner

b FAST barrier

Returns:

0 for non-corner, minimum increment required to make detector go down different branch, if it is a corner.

Definition at line 99 of file faster_bytecode.h.

References d.

Referenced by tree_detect_corners(), and tree_detect_corners_all().

```
100     {
101         int n=0;
102         int m = INT_MAX;
103         int cb = *imp + b;
104         int c_b = *imp - b;
105         int p;
106 
107         while(d[n].lt)
108         {
109             p = imp[d[n].offset];
110 
111             if(p > cb)
112             {
113                 if(p-cb < m)
114                     m = p-cb;
115 
116                 n = d[n].gt;
117             }
118             else if(p < c_b)
119             {
120                 if(c_b - p < m)
121                     m = c_b - p;
122 
123                 n = d[n].lt;
124             }
125             else
126                 n = d[n].eq;
127         }
128 
129         if(d[n].gt)
130             return m;
131         else
132             return 0;
133     }
```

7.1.2.3 void block_bytecode::print (std::ostream & *o*, int *width*) const [inline]

Serialize the detector to an ostream.

The serialized detector a number of lines of the form:

```
Block N [X Y] G E L
```

or:

```
Block N corner
```

or:

```
Block N non_corner
```

The first block type represents the code:

```
if Image[current_pixel + (x, y)] > Image[current_pixel] + threshold
   goto block G
elseif Image[current_pixel + (x, y)] < Image[current_pixel] -threshold
   goto block L
else
   goto block E
endif
```

Parameters:

o ostream for output

width width the detector was created at, required to back out the offsets correctly.

Definition at line 160 of file faster_bytecode.h.

References d.

Referenced by faster_learn::faster_learn(), and run_learn_detector().

```
161     {
162         using tag::operator<<;
163         for(unsigned int i=0; i < d.size(); i++)
164         {
165             if(d[i].lt == 0)
166                 o << tag::print << "Block" << i <<  (d[i].gt?"corner":"non_corner");
167             else
168             {
169                 int a = abs(d[i].offset) + width / 2;
170                 if(d[i].offset < 0)
171                     a = -a;
172                 int y = a / width;
173 
174                 int x = d[i].offset - y * width;
175                 o << tag::print << "Block" << i << CVD::ImageRef(x , y) << d[i].gt << d[i].eq << d[i].
176             }
177         }
178     }
```

7.1.2.4 void block_bytecode::detect (const CVD::Image< CVD::byte > & *im*, std::vector< int > & *corners*, int *threshold*, int *xmin*, int *xmax*, int *ymin*, int *ymax*)

Detect corners in an image.

The width of the image must match the width the detector was compiled to (using tree_elemeent::make_-fast_detector for the results to make sense. The bytecode is JIT coimpiled if possible.

Parameters:

im The image in which to detect corners

corners Detected corners are inserted in to this container.

threshold Corner detector threshold to use

xmin x coordinate to start at.

ymin y coordinate to start at.

xmax x coordinate to go up to.

ymax y coordinate to go up to.

Definition at line 329 of file faster_bytecode.cc.

References d, and detect_no_score().

```
330 {
331     #ifdef JIT
332         jit_detector jit(d);
333         for(int y = ymin; y < ymax; y++)
334             jit.detect_in_row(im, y, xmin, xmax, corners, threshold);
335     #else
```

```
336         cerr << "Hello!\n";
337         for(int y = ymin; y < ymax; y++)
338             for(int x=xmin; x < xmax; x++)
339                 if(detect_no_score(&im[y][x], threshold))
340                     corners.push_back(&im[y][x] - im.data());
341     #endif
342 }
```

7.1.3 Member Data Documentation

7.1.3.1 std::vector<fast_detector_bit> block_bytecode::d

This contains the compiled bytecode.

Definition at line 64 of file faster_bytecode.h.

Referenced by detect(), detect_no_score(), and print().

The documentation for this struct was generated from the following files:

- faster_bytecode.h
- faster_bytecode.cc

7.2 block_bytecode::fast_detector_bit Struct Reference

`#include <faster_bytecode.h>`

7.2.1 Detailed Description

This is a bytecode element for the bytecode-compiled detector.

The bytecode consists of a number of fixed length blocks representing a 3 way branch. Special values of of a block indicate the result that a pixel is a corner or non-corner.

Specifically, if `lt == 0`, then this is a leaf and `gt` holds the class. The root node is always stored as the first bytecode instruction.

Definition at line 48 of file faster_bytecode.h.

Public Attributes

- int offset
- int lt
- int gt
- int eq

7.2.2 Member Data Documentation

7.2.2.1 int block_bytecode::fast_detector_bit::offset

Memory offset from centre pixel to examine.

This means that the fast detector must be created for an image of a known width.

Definition at line 50 of file faster_bytecode.h.

7.2.2.2 int block_bytecode::fast_detector_bit::lt

Position in bytecode to branch to if offset pixel is much darker than the centre pixel.

If this is zero, then gt stores the result.

Definition at line 56 of file faster_bytecode.h.

7.2.2.3 int block_bytecode::fast_detector_bit::gt

Position in bytecode to branch to if offset pixel is much brighter than the centre pixel.

If lt==0 is a result block, then this stores the result, 0 for a non corner, 1 for a corner.

Definition at line 58 of file faster_bytecode.h.

7.2.2.4 int block_bytecode::fast_detector_bit::eq

Position in bytecode to branch to otherwise.

Definition at line 60 of file faster_bytecode.h.

The documentation for this struct was generated from the following file:

- faster_bytecode.h

7.3 datapoint< FEATURE_SIZE > Struct Template Reference

7.3.1 Detailed Description

template<int FEATURE_SIZE> struct datapoint< FEATURE_SIZE >

This structure represents a datapoint.

A datapoint is a group of pixels with ternary values (much brighter than the centre, much darker than the centre or similar to the centre pixel). In addition to the feature descriptor, the class and number of instances is also stored.

The maximum feature vector size is determined by the template parameter. This allows the ternary vector to be stored in a bitset. This keeps the struct a fixed size and removes the need for dynamic allocation.

Definition at line 146 of file learn_fast_tree.cc.

Public Member Functions

- datapoint (const string &s, unsigned long c, bool is)
- datapoint ()
- Ternary get_trit (unsigned int tnum) const

Public Attributes

- unsigned long count
- bool is_a_corner

Static Public Attributes

- static const unsigned int max_size = FEATURE_SIZE

Private Member Functions

- void pack_trits (const string &unpacked)
- void set_trit (unsigned int tnum, Ternary val)

Private Attributes

- bitset< max_size *2 > tests

7.3.2 Constructor & Destructor Documentation

7.3.2.1 template<int FEATURE_SIZE> datapoint< FEATURE_SIZE >::datapoint (const string & *s*, unsigned long *c*, bool *is*) [inline]

Construct a datapoint.

Parameters:

s The feature vector in string form

c The number of instances

is The class

Definition at line 152 of file learn_fast_tree.cc.

References datapoint< FEATURE_SIZE >::pack_trits().

```
153     :count(c),is_a_corner(is)
154     {
155         pack_trits(s);
156     }
```

7.3.2.2 template<int FEATURE_SIZE> datapoint< FEATURE_SIZE >::datapoint () [inline]

Default constructor allows for storage in a std::vector.

Definition at line 160 of file learn_fast_tree.cc.

```
161     {}
```

7.3.3 Member Function Documentation

7.3.3.1 template<int FEATURE_SIZE> Ternary datapoint< FEATURE_SIZE >::get_trit (unsigned int *tnum*) const [inline]

Extract a trit (ternary bit) from the feture vector.

Parameters:

tnum Number of the bit to extract

Returns:

The trit.

Definition at line 172 of file learn_fast_tree.cc.

References Brighter, Darker, datapoint< FEATURE_SIZE >::max_size, Similar, and datapoint< FEATURE_SIZE >::tests.

```
173     {
174         assert(tnum < size);
175         if(tests[tnum] == 1)
176             return Brighter;
177         else if(tests[tnum + max_size] == 1)
178             return Darker;
179         else
180             return Similar;
181     }
```

7.3.3.2 template<int FEATURE_SIZE> void datapoint< FEATURE_SIZE >::pack_trits (const string & *unpacked*) [inline, private]

This code reads a stringified representation of the feature vector and converts it in to the internal representation.

The string represents one feature per character, using "b", "d" and "s".

Parameters:

unpacked String to parse.

Definition at line 203 of file learn_fast_tree.cc.

References Brighter, Darker, fatal, datapoint< FEATURE_SIZE >::set_trit(), Similar, and datapoint< FEATURE_SIZE >::tests.

Referenced by datapoint< FEATURE_SIZE >::datapoint().

```
204         {
205             tests = 0;
206             for(unsigned int i=0;i < unpacked.size(); i++)
207             {
208                 if(unpacked[i] == 'b')
209                     set_trit(i, Brighter);
210                 else if(unpacked[i] == 'd')
211                     set_trit(i, Darker);
212                 else if(unpacked[i] == 's')
213                     set_trit(i, Similar);
214                 else
215                     fatal(2, "Bad char while packing datapoint: %s", unpacked);
216             }
217         }
```

7.3.3.3 template<int FEATURE_SIZE> void datapoint< FEATURE_SIZE >::set_trit (unsigned int *tnum*, Ternary *val*) [inline, private]

Set a ternary digit.

Parameters:

tnum Digit to set

val Value to set it to.

Definition at line 222 of file learn_fast_tree.cc.

References Brighter, Darker, datapoint< FEATURE_SIZE >::max_size, Similar, and datapoint< FEATURE_SIZE >::tests.

Referenced by datapoint< FEATURE_SIZE >::pack_trits().

```
223         {
224             assert(val == Brighter || val == Darker || val == Similar);
225             assert(tnum < max_size);
226 
227             if(val == Brighter)
228                 tests[tnum] = 1;
229             else if(val == Darker)
230                 tests[tnum + max_size] = 1;
231         }
```

7.3.4 Member Data Documentation

7.3.4.1 template<int FEATURE_SIZE> unsigned long datapoint< FEATURE_SIZE >::count

Number of instances.

Definition at line 163 of file learn_fast_tree.cc.

7.3.4.2 template<int FEATURE_SIZE> bool datapoint< FEATURE_SIZE >::is_a_corner

Class.

Definition at line 164 of file learn_fast_tree.cc.

7.3.4.3 template<int FEATURE_SIZE> const unsigned int datapoint< FEATURE_SIZE >::max_size = FEATURE_SIZE [static]

Maximum number of features representable.

Definition at line 166 of file learn_fast_tree.cc.

Referenced by datapoint< FEATURE_SIZE >::get_trit(), and datapoint< FEATURE_SIZE >::set_trit().

7.3.4.4 template<int FEATURE_SIZE> bitset<max_size*2> datapoint< FEATURE_SIZE >::tests [private]

Used to store the ternary vector Ternary bits are stored using 3 out of the 4 values storable by two bits.

Trit *n* is stored using the bits *n* and *n* + *max_size*, with bit *n* being the most significant bit.

The values are

- 3 unused
- 2 Brighter
- 1 Darker
- 0 Similar

Definition at line 185 of file learn_fast_tree.cc.

Referenced by datapoint< FEATURE_SIZE >::get_trit(), datapoint< FEATURE_SIZE >::pack_trits(), and datapoint< FEATURE_SIZE >::set_trit().

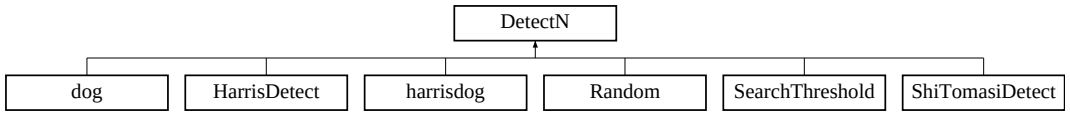
The documentation for this struct was generated from the following file:

- learn_fast_tree.cc

7.4 DetectN Struct Reference

`#include <detectors.h>`

Inheritance diagram for DetectN::



7.4.1 Detailed Description

A corner detector object which is passed a target number of corners to detect.

Definition at line 31 of file detectors.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const =0
- virtual ~DetectN ()

7.4.2 Constructor & Destructor Documentation

7.4.2.1 virtual DetectN::~DetectN () [inline, virtual]

Destroy to object.

Definition at line 39 of file detectors.h.

```
39 {}
```

7.4.3 Member Function Documentation

7.4.3.1 virtual void DetectN::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [pure virtual]

Detect corners.

Parameters:

- ***i*** Image in which to detect corners
- ***c*** Detected corners are inserted in to this container
- ***N*** Number of corners to detect

Implemented in dog, harrisdog, ShiTomasiDetect, and HarrisDetect.

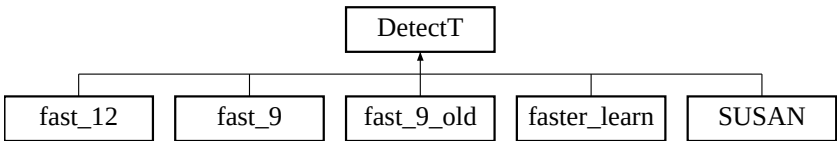
The documentation for this struct was generated from the following file:

- detectors.h

7.5 DetectT Struct Reference

`#include <detectors.h>`

Inheritance diagram for DetectT::



7.5.1 Detailed Description

A corner detector object which is passed a threshold.

These can be wrapped with a searching algorithm to turn them in to a DetectN, specifically SearchThreshold

Definition at line 45 of file detectors.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const =0
- virtual ~DetectT ()

7.5.2 Constructor & Destructor Documentation

7.5.2.1 virtual DetectT::~DetectT () `[inline, virtual]`

Destroy to object.

Definition at line 53 of file detectors.h.

```
53 {}
```

7.5.3 Member Function Documentation

7.5.3.1 virtual void DetectT::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const `[pure virtual]`

Detect corners.

Parameters:

- ***i*** Image in which to detect corners
- ***c*** Detected corners are inserted in to this container
- ***N*** Threshold used to detect corners

Implemented in fast_9, fast_9_old, fast_12, faster_learn, and SUSAN.

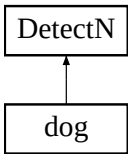
The documentation for this struct was generated from the following file:

- detectors.h

7.6 dog Struct Reference

`#include <dog.h>`

Inheritance diagram for dog::



7.6.1 Detailed Description

Class wrapping the Difference of Gaussians detector.

Definition at line 32 of file dog.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.6.2 Member Function Documentation

7.6.2.1 void dog::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Number of corners to detect

Implements DetectN.

Definition at line 109 of file dog.cc.

```
110 {
111     int s = GV3::get<int>("dog.divisions_per_octave", 3,1); //Divisions per octave
112     int octaves=GV3::get<int>("dog.octaves", 4, 1);
113
114     double k = pow(2, 1.0/s);
115
116     double sigma = GV3::get<double>("dog.sigma", 0.8, 1);
117
118     Image<float> im = convert_image(i);
119
120     convolveGaussian_fir(im, im, sigma);
121
122
123     Image<float> d1, d2, d3;
```

```
124     c.clear();
125     vector<pair<float, ImageRef> > corners;
126     corners.reserve(50000);
127 
128     int scalemul=1;
129     int d1m = 1, d2m = 1, d3m = 1;
130 
131     for(int o=0; o < octaves; o++)
132     {
133 
134         for(int j=0; j < s; j++)
135         {
136             float delta_sigma = sigma * sqrt(k*k-1);
137             Image<float> blurred(im.size());
138             convolveGaussian_fir(im, blurred, delta_sigma);
139 
140             for(fi i1=im.begin(), i2 = blurred.begin(); i1!= im.end(); ++i1, ++i2)
141                 *i1 = (*i2 - *i1);
142 
143             //im is now dog
144             //blurred
145 
146             d1 = d2;
147             d2 = d3;
148             d3 = im;
149             im = blurred;
150 
151             d1m = d2m;
152             d2m = d3m;
153             d3m = scalemul;
154 
155             //Find maxima
156             if(d1.size().x != 0)
157             {
158                 if(d1.size() == d2.size())
159                     if(d2.size() == d3.size())
160                         local_maxima<Equal, Equal>(d1, d2, d3, corners, d2m);
161                     else
162                         local_maxima<Equal, Smaller>(d1, d2, d3, corners, d2m);
163                 else
164                     if(d2.size() == d3.size())
165                         local_maxima<Larger, Equal>(d1, d2, d3, corners, d2m);
166                     else
167                         local_maxima<Larger, Smaller>(d1, d2, d3, corners, d2m);
168             }
169 
170 
171             sigma *= k;
172         }
173 
174         if(o != octaves - 1)
175         {
176             scalemul *=2;
177             sigma /=2;
178             Image<float> tmp(im.size()/2);
179             halfSample(im,tmp);
180             im=tmp;
181         }
182     }
183 
184 
185     if(corners.size() > N)
186     {
187         nth_element(corners.begin(), corners.begin() + N, corners.end());
188         corners.resize(N);
189     }
190 
```

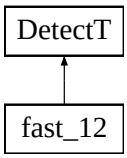
```
191
192     for(unsigned int i=0; i < corners.size(); i++)
193         c.push_back(corners[i].second);
194 }
```

The documentation for this struct was generated from the following files:

- dog.h
- dog.cc

7.7 fast_12 Struct Reference

Inheritance diagram for fast_12::



7.7.1 Detailed Description

Definition at line 40 of file cvd_fast.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.7.2 Member Function Documentation

7.7.2.1 void fast_12::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Threshold used to detect corners

Implements DetectT.

Definition at line 50 of file cvd_fast.cc.

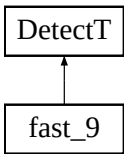
```
51 {
52     vector<ImageRef> cs;
53     fast_corner_detect_12(i, cs, n);
54     vector<int> sc;
55     fast_corner_score_12(i, cs, n, sc);
56     nonmax_suppression(cs, sc, c);
57 }
```

The documentation for this struct was generated from the following files:

- cvd_fast.h
- cvd_fast.cc

7.8 fast_9 Struct Reference

Inheritance diagram for fast_9::



7.8.1 Detailed Description

Definition at line 30 of file cvd_fast.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.8.2 Member Function Documentation

7.8.2.1 void fast_9::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Threshold used to detect corners

Implements DetectT.

Definition at line 45 of file cvd_fast.cc.

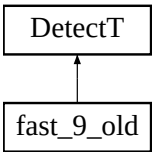
```
46 {
47     fast_corner_detect_9_nonmax(i, c, static_cast<int>(n));
48 }
```

The documentation for this struct was generated from the following files:

- cvd_fast.h
- cvd_fast.cc

7.9 fast_9_old Struct Reference

Inheritance diagram for fast_9_old::



7.9.1 Detailed Description

Definition at line 35 of file cvd_fast.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.9.2 Member Function Documentation

7.9.2.1 void fast_9_old::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Threshold used to detect corners

Implements DetectT.

Definition at line 37 of file cvd_fast.cc.

```
38 {
39     vector<ImageRef> ct;
40     vector<int> sc;
41     fast_corner_detect_9(i, ct, static_cast<int>(n));
42     fast_nonmax(i, ct, static_cast<int>(n), c);
43 }
```

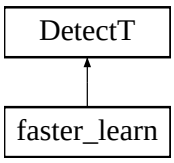
The documentation for this struct was generated from the following files:

- cvd_fast.h
- cvd_fast.cc

7.10 faster_learn Struct Reference

`#include <faster_detector.h>`

Inheritance diagram for faster_learn::



7.10.1 Detailed Description

FAST-ER detector.

Definition at line 36 of file faster_detector.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const
- faster_learn (const std::string &fname)

Private Attributes

- std::auto_ptr< tree_element > tree

7.10.2 Constructor & Destructor Documentation

7.10.2.1 faster_learn::faster_learn (const std::string & *fname*)

Initialize a detector.

Parameters:

fname File to load the detector from. This was created from learn_detector.

Definition at line 72 of file faster_detector.cc.

References load_a_tree(), and block_bytecode::print().

```
73 {
74     init();
75     ifstream i;
76     i.open(fname.c_str());
77 
78     if(!i.good())
79     {
80         cerr << "Error: " << fname << ": " << strerror(errno) << endl;
81         exit(1);
82     }
83 
```

```
84      try{
85          tree.reset(load_a_tree(i));
86      }
87      catch(ParseError p)
88      {
89          cerr << "Parse error in " << fname << endl;
90          exit(1);
91      }
92
93      if(GV3::get<bool>("faster_tree.print_tree", 0, 1))
94      {
95          clog << "Tree:" << endl;
96          tree->print(clog);
97      }
98
99      if(GV3::get<bool>("faster_tree.print_block", 0, 1))
100      {
101          block_bytecode f2 = tree->make_fast_detector(100);
102          f2.print(clog,  100);
103      }
104 }
```

7.10.3 Member Function Documentation

7.10.3.1 void faster_learn::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

- ***i*** Image in which to detect corners
- ***c*** Detected corners are inserted in to this container
- ***N*** Threshold used to detect corners

Implements DetectT.

Definition at line 107 of file faster_detector.cc.

References tree_detect_corners().

```
108 {
109     Image<int> scratch(i.size(), 0);
110
111     v = tree_detect_corners(i, tree.get(), t, scratch);
112 }
```

7.10.4 Member Data Documentation

7.10.4.1 std::auto_ptr<tree_element> faster_learn::tree [private]

Loaded FAST-ER tree.

Definition at line 50 of file faster_detector.h.

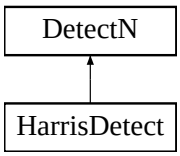
The documentation for this struct was generated from the following files:

- faster_detector.h
- faster_detector.cc

7.11 HarrisDetect Struct Reference

`#include <harrislike.h>`

Inheritance diagram for HarrisDetect::



7.11.1 Detailed Description

Class wrapping the Shi-Tomasi detector.

Definition at line 43 of file harrislike.h.

Public Member Functions

- void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.11.2 Member Function Documentation

7.11.2.1 void HarrisDetect::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Number of corners to detect

Implements DetectN.

Definition at line 157 of file harrislike.cc.

```
158 {
159     float blur = GV3::get<float>("harris.blur", 2.5, 1);
160     float sigmas = GV3::get<float>("harris.sigmas", 2.0, 1);
161     harris_like<HarrisScore,PosInserter>(i, c, N, blur, sigmas);
162 }
```

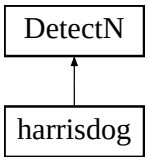
The documentation for this struct was generated from the following files:

- harrislike.h
- harrislike.cc

7.12 harrisdog Struct Reference

`#include <dog.h>`

Inheritance diagram for harrisdog::



7.12.1 Detailed Description

Class wrapping the Harris-Laplace detector.

Definition at line 43 of file dog.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.12.2 Member Function Documentation

7.12.2.1 void harrisdog::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detecto corners.

Parameters:

- ***i*** Image in which to detect corners
- ***c*** Detected corners are inserted in to this container
- ***N*** Number of corners to detect

Implements DetectN.

Definition at line 226 of file dog.cc.

References HarrisDetector().

```
227 {
228     int s = GV3::get<int>("harrislaplace.dog.divisions_per_octave", 11);    //Divisions per octave
229     int octaves=GV3::get<int>("harrislaplace.dog.octaves", 4, 1);
230     double sigma = GV3::get<double>("harrislaplace.dog.sigma", 0.8);
231
232     float hblur = GV3::get<float>("harrislaplace.harris.blur", 2.5);
233     float hsigmas = GV3::get<float>("harrislaplace.harris.sigmas", 2.0, 1);
234
235     double k = pow(2, 1.0/s);
236
237     Image<float> im = convert_image(i);
238
```

```
239     //convolveGaussian(im, sigma);
240
241     Image<float> d1, d2, d3;
242     Image<float> im1, im2, im3;
243     c.clear();
244     vector<pair<float, ImageRef> > corners;
245     corners.reserve(50000);
246
247     int scalemul=1;
248     int d1m = 1, d2m = 1, d3m = 1;
249
250     for(int o=0; o < octaves; o++)
251     {
252
253         for(int j=0; j < s; j++)
254         {
255             float delta_sigma = sigma * sqrt(k*k-1);
256             //hblur *= sqrt(k*k-1);
257
258             //Blur im, and put the result in blurred.
259             //im is already blurred from the previous layers
260             Image<float> blurred(im.size(), 0);
261             convolveGaussian_fir(im, blurred, delta_sigma);
262
263             //For DoG, at this point, we don't need im anymore, since blurred
264             //will be used as "im" for the next layer. However, we do need it for
265             //HarrisDoG, since we need to do a HarrisDetect on it.
266             Image<float>  diff(im.size(), 0);
267             for(fi i1=im.begin(), i2 = blurred.begin(), d = diff.begin(); i1!= im.end(); ++i1, ++i2, +
268                 *d = (*i2 - *i1);
269
270             //Insert the current image, and the current difference
271             //in to the ring buffer
272             d1 = d2;
273             d2 = d3;
274             d3 = diff;
275
276             im1 = im2;
277             im2 = im3;
278             im3 = im;
279
280
281             im = blurred;
282
283             d1m = d2m;
284             d2m = d3m;
285             d3m = scalemul;
286
287             //Find maxima
288             if(d1.size().x != 0)
289             {
290                 //First, find Harris maxima
291                 vector<pair<float, ImageRef> > layer_corners;
292                 HarrisDetector(im2, layer_corners, N, hblur, hsigmas);
293
294                 //Keep if they are LoG (or really DoG) maxima across scales.
295                 //The Harris score olny is used.
296                 for(unsigned int c=0; c < layer_corners.size(); c++)
297                     if(is_scale_maximum(d1, d2, d3, layer_corners[c].second))
298                         corners.push_back(layer_corners[c]);
299             }
300
301             sigma *= k;
302         }
303
304         if(o != octaves - 1)
305         {
```

```
306             scalemul *=2;
307             sigma /=2;
308             Image<float> tmp(im.size()/2);
309             halfSample(im,tmp);
310             im=tmp;
311         }
312     }
313 
314 
315     if(corners.size() > N)
316     {
317         nth_element(corners.begin(), corners.begin() + N, corners.end());
318         corners.resize(N);
319     }
320 
321     c.clear();
322 
323     for(unsigned int i=0; i < corners.size(); i++)
324         c.push_back(corners[i].second);
325 
326 }
```

The documentation for this struct was generated from the following files:

- dog.h
- dog.cc

7.13 ParseError Struct Reference

`#include <faster_tree.h>`

7.13.1 Detailed Description

A named symbol to throw in the case that tree deserialization fails with a parse error.

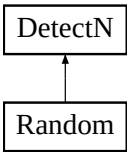
Definition at line 398 of file faster_tree.h.

The documentation for this struct was generated from the following file:

- faster_tree.h

7.14 Random Struct Reference

Inheritance diagram for Random::



7.14.1 Detailed Description

Detector which randomly scatters corners around an image.

Definition at line 127 of file detectors.cc.

Public Member Functions

- virtual void operator() (const Image< byte > &im, vector< ImageRef > &corners, unsigned int N) const
- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const =0

7.14.2 Member Function Documentation

7.14.2.1 virtual void Random::operator() (const Image< byte > & *im*, vector< ImageRef > & *corners*, unsigned int *N*) const [inline, virtual]

Detect corners by scattering points around at random.

Parameters:

im Image in which to detect corners

corners Detected corners are inserted in to this array

N number of corners to detect

Definition at line 133 of file detectors.cc.

```
134     {
135         for(unsigned int i=0; i < N; i++)
136             corners.push_back(ImageRef(rand() % im.size().x, rand() % im.size().y));
137     }
```

7.14.2.2 virtual void DetectN::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [pure virtual, inherited]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Number of corners to detect

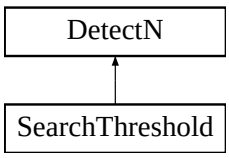
Implemented in dog, harrisdog, ShiTomasiDetect, and HarrisDetect.

The documentation for this struct was generated from the following file:

- detectors.cc

7.15 SearchThreshold Struct Reference

Inheritance diagram for SearchThreshold::



7.15.1 Detailed Description

This class wraps a DetectT class with binary_search_threshold and presents is as a DetectN class.

Definition at line 103 of file detectors.cc.

Public Member Functions

- SearchThreshold (DetectT *d)
- virtual void operator() (const Image< byte > &im, vector< ImageRef > &corners, unsigned int N) const
- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const =0

Private Attributes

- auto_ptr< DetectT > detector

7.15.2 Constructor & Destructor Documentation

7.15.2.1 SearchThreshold::SearchThreshold (DetectT * *d*) [inline]

Parameters:

d Detector to wrap. This will be managed by SearchThreshold

Definition at line 106 of file detectors.cc.

```
107     :detector(d)
108     {
109     }
```

7.15.3 Member Function Documentation

7.15.3.1 virtual void SearchThreshold::operator() (const Image< byte > & *im*, vector< ImageRef > & *corners*, unsigned int *N*) const [inline, virtual]

Detect corners.

Parameters:

im Image in which to detect corners

corners Detected corners are inserted in to this array

N number of corners to detect

Definition at line 115 of file detectors.cc.

References binary_search_threshold(), and detector.

```
116     {
117         int t = binary_search_threshold(im, corners, N, *detector);
118     }
```

7.15.3.2 virtual void DetectN::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [pure virtual, inherited]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Number of corners to detect

Implemented in dog, harrisdog, ShiTomasiDetect, and HarrisDetect.

7.15.4 Member Data Documentation

7.15.4.1 auto_ptr<DetectT> SearchThreshold::detector [private]

Detector to wrap.

Definition at line 122 of file detectors.cc.

Referenced by operator()().

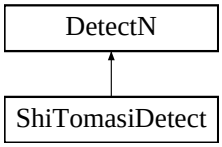
The documentation for this struct was generated from the following file:

- detectors.cc

7.16 ShiTomasiDetect Struct Reference

`#include <harrislike.h>`

Inheritance diagram for ShiTomasiDetect::



7.16.1 Detailed Description

Class wrapping the Harris detector.

Definition at line 31 of file harrislike.h.

Public Member Functions

- void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.16.2 Member Function Documentation

7.16.2.1 void ShiTomasiDetect::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

- ***i*** Image in which to detect corners
- ***c*** Detected corners are inserted in to this container
- ***N*** Number of corners to detect

Implements DetectN.

Definition at line 164 of file harrislike.cc.

```
165 {
166     float blur = GV3::get<float>("shitomasi.blur", 2.5, 1);
167     float sigmas = GV3::get<float>("shitomasi.sigmas", 2.0, 1);
168     harris_like<ShiTomasiScore, PosInserter>(i, c, N, blur, sigmas);
169 }
```

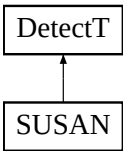
The documentation for this struct was generated from the following files:

- harrislike.h
- harrislike.cc

7.17 SUSAN Struct Reference

`#include <susan.h>`

Inheritance diagram for SUSAN::



7.17.1 Detailed Description

Class wrapping the SUSAN detector.

Definition at line 32 of file susan.h.

Public Member Functions

- virtual void operator() (const CVD::Image< CVD::byte > &i, std::vector< CVD::ImageRef > &c, unsigned int N) const

7.17.2 Member Function Documentation

7.17.2.1 void SUSAN::operator() (const CVD::Image< CVD::byte > & *i*, std::vector< CVD::ImageRef > & *c*, unsigned int *N*) const [virtual]

Detect corners.

Parameters:

i Image in which to detect corners

c Detected corners are inserted in to this container

N Threshold used to detect corners

Implements DetectT.

Definition at line 48 of file susan.cc.

```
49 {
50     float dt = GV3::get<float>("susan.dt", 4.0, 1);
51     int* c = susan(const_cast<byte*>(im.data()), im.size().x, im.size().y, dt, N);
52
53     int n = c[0];
54
55     for(int i=0; i < n; i++)
56         corners.push_back(ImageRef(c[2*i+2], c[2*i+3]));
57
58     free_haxored_memory();
59 }
```

The documentation for this struct was generated from the following files:

- susan.h
- susan.cc

7.18 tree Struct Reference

7.18.1 Detailed Description

This class represents a decision tree.

Each leaf node contains a class, being Corner or NonCorner. Each decision node contains a feature about which to make a ternary decision. Additionally, each node records how many datapoints were tested. The generated tree structure is not mutable.

Definition at line 387 of file learn_fast_tree.cc.

Public Types

- enum IsCorner { Corner, NonCorner, NonTerminal }

Public Member Functions

- string stringify ()
- tree (shared_ptr< tree > b, shared_ptr< tree > d, shared_ptr< tree > s, int n, uint64_t num)

Static Public Member Functions

- static shared_ptr< tree > CornerLeaf (uint64_t n)
- static shared_ptr< tree > NonCornerLeaf (uint64_t n)

Public Attributes

- const shared_ptr< tree > brighter
- const shared_ptr< tree > darker
- const shared_ptr< tree > similar
- const IsCorner is_a_corner
- const int feature_to_test
- const uint64_t num_datapoints

Private Member Functions

- tree (IsCorner c, uint64_t n)

7.18.2 Member Enumeration Documentation

7.18.2.1 enum tree::IsCorner

The class of the leaf, and a sentinal to indacate that the node is not a leaf.

Now that I come back to this, it looks suspiciously like an instance of http://thedailywtf.com/Articles/What_Is_Truth_0x3f_.aspx Oh well.

Enumerator:

Corner

NonCorner

NonTerminal

Definition at line 392 of file learn_fast_tree.cc.

```
393     {
394         Corner,
395         NonCorner,
396         NonTerminal
397     };
```

7.18.3 Constructor & Destructor Documentation

7.18.3.1 tree::tree (shared_ptr< tree > *b*, shared_ptr< tree > *d*, shared_ptr< tree > *s*, int *n*, uint64_t *num*) [inline]

Create a non-leaf node.

Parameters:

b The brighter subtree

d The darker subtree

s The similar subtree

n Feature number to test

num Number of datapoints reaching this node.

Definition at line 443 of file learn_fast_tree.cc.

Referenced by CornerLeaf(), and NonCornerLeaf().

```
444     :brighter(b), darker(d), similar(s), is_a_corner(NonTerminal), feature_to_test(n), num_datapoints(
445     {}
```

7.18.3.2 tree::tree (IsCorner *c*, uint64_t *n*) [inline, private]

The leaf node constructor is private to prevent a tree being constructed with invalid values.

see also CornerLeaf and NonCornerLeaf.

Parameters:

c Class of the node

n Number of datapoints which this node represents

Definition at line 453 of file learn_fast_tree.cc.

```
454     :is_a_corner(c),feature_to_test(-1),num_datapoints(n)
455     {}
```

7.18.4 Member Function Documentation

7.18.4.1 string tree::stringify () [inline]

Convert the tree to a simple string representation.

This is allows comparison of two trees to see if they are the same. It's probably rather inefficient to hammer the string class compared to using an ostringstream, but this is not the slowest part of the program.

Returns:

a stringified tree representation

Definition at line 411 of file learn_fast_tree.cc.

References brighter, Corner, darker, is_a_corner, NonTerminal, and similar.

```
412     {
413         if(is_a_corner == NonTerminal)
414             return "(" + brighter->stringify() + darker->stringify() + similar->stringify() + ")";
415         else
416             return string("(") + (is_a_corner == Corner?"1":"0")  +  ")";
417     }
```

7.18.4.2 static shared_ptr<tree> tree::CornerLeaf (uint64_t *n*) [inline, static]

Create a leaf node which is a corner This special constructor function makes it impossible to construct a leaf with the NonTerminal class.

Parameters:

n number of datapoints reaching this node.

Definition at line 423 of file learn_fast_tree.cc.

References Corner, and tree().

Referenced by build_tree().

```
424     {
425         return shared_ptr<tree>(new tree(Corner, n));
426     }
```

7.18.4.3 static shared_ptr<tree> tree::NonCornerLeaf (uint64_t *n*) [inline, static]

Creat a leaf node which is a non-corner This special constructor function makes it impossible to construct a leaf with the NonTerminal class.

Parameters:

n number of datapoints reaching this node.

Definition at line 432 of file learn_fast_tree.cc.

References NonCorner, and tree().

Referenced by build_tree().

```
433     {
434         return shared_ptr<tree>(new tree(NonCorner, n));
435     }
```

7.18.5 Member Data Documentation

7.18.5.1 const shared_ptr<tree> tree::brighter

Subtrees.

Definition at line 399 of file learn_fast_tree.cc.

Referenced by print_tree(), and stringify().

7.18.5.2 const shared_ptr<tree> tree::darker

Subtrees.

Definition at line 400 of file learn_fast_tree.cc.

Referenced by print_tree(), and stringify().

7.18.5.3 const shared_ptr<tree> tree::similar

Subtrees.

Definition at line 401 of file learn_fast_tree.cc.

Referenced by print_tree(), and stringify().

7.18.5.4 const IsCorner tree::is_a_corner

Class of this node (if its a leaf).

Definition at line 402 of file learn_fast_tree.cc.

Referenced by print_tree(), and stringify().

7.18.5.5 const int tree::feature_to_test

Feature (ie pixel) to test if this is a non-leaf.

Definition at line 403 of file learn_fast_tree.cc.

Referenced by print_tree().

7.18.5.6 const uint64_t tree::num_datapoints

Number of datapoints passing through this node.

Definition at line 404 of file learn_fast_tree.cc.

Referenced by print_tree().

The documentation for this struct was generated from the following file:

- learn_fast_tree.cc

7.19 tree_element Class Reference

`#include <faster_tree.h>`

7.19.1 Detailed Description

This struct represents a node of the tree, and has pointers to other structs, thereby representing a branch or the entire tree.

Definition at line 39 of file faster_tree.h.

Public Member Functions

- std::pair< CVD::ImageRef, CVD::ImageRef > bbox () const
- int num_nodes () const
- bool is_leaf () const
- std::pair< tree_element *, bool > nth_element (int t)
- block_bytecode make_fast_detector (int xsize) const
- int detect_corner (const CVD::Image< CVD::byte > &im, CVD::ImageRef pos, int b) const
- tree_element * copy ()
- void print (std::ostream &o, std::string ind=" ") const
- ~tree_element ()
- tree_element (bool b)
- tree_element (tree_element *a, tree_element *b, tree_element *c, int i)

Public Attributes

- tree_element * lt
- tree_element * eq
- tree_element * gt
- bool is_corner
- int offset_index

Private Member Functions

- void make_fast_detector_o (std::vector< block_bytecode::fast_detector_bit > &v, int n, int xsize, int N, bool invert) const
- std::pair< tree_element *, bool > nth_element (int target, int &n, bool eq_branch)
- int detect_corner_oriented (const CVD::Image< CVD::byte > &im, CVD::ImageRef pos, int b, int n, bool invert) const

7.19.2 Constructor & Destructor Documentation

7.19.2.1 tree_element::~tree_element () [inline]

Destruct the tree node.

This destructs all child nodes, so deleting a tree a deep modification operation.

Definition at line 366 of file faster_tree.h.

```
367         {
368             delete lt;
369             delete eq;
370             delete gt;
371         }
```

7.19.2.2 tree_element::tree_element (bool *b*) [inline]

Construct a leaf-node.

Parameters:

b Class of the node

Definition at line 375 of file faster_tree.h.

```
376         :lt(0),eq(0),gt(0),is_corner(b),offset_index(0)
377         {}
```

7.19.2.3 tree_element::tree_element (tree_element * *a*, tree_element * *b*, tree_element * *c*, int *i*) [inline]

Construct a non-leaf tree node.

Parameters:

a Less-Than branch of tree

b Equal branch of tree

c Greater-Than branch of tree

i Pixel number to examine.

Definition at line 384 of file faster_tree.h.

```
385         :lt(a),eq(b),gt(c),is_corner(0),offset_index(i)
386         {}
```

7.19.3 Member Function Documentation

7.19.3.1 std::pair<CVD::ImageRef, CVD::ImageRef> tree_element::bbox () const [inline]

This returns the bounding box of the detector.

Definition at line 50 of file faster_tree.h.

References offsets_bbox.

Referenced by tree_detect_corners(), and tree_detect_corners_all().

```
51         {
52             return offsets_bbox;
53         }
```

7.19.3.2 int tree_element::num_nodes () const [inline]

This returns the number of nodes in the tree.

Definition at line 56 of file faster_tree.h.

References eq, gt, lt, and num_nodes().

Referenced by learn_detector(), and num_nodes().

```
57         {
58             if(eq == NULL)
59                 return 1;
60             else
61                 return 1 + lt->num_nodes() + eq->num_nodes() + gt->num_nodes();
62         }
```

7.19.3.3 bool tree_element::is_leaf () const [inline]

Is the node a leaf?

Definition at line 66 of file faster_tree.h.

References eq.

Referenced by learn_detector(), and make_fast_detector_o().

```
67         {
68             return eq == NULL;
69         }
```

7.19.3.4 std::pair<tree_element*,bool> tree_element::nth_element (int *t*) [inline]

Return a given numbered element of the tree.

Elements are numbered by depth-first traversal.

Parameters:

t Element number to return

Returns:

pointer to the t'th element, and a flag indicating whether it's the direct child of an eq branch.

Definition at line 76 of file faster_tree.h.

Referenced by learn_detector().

```
77         {
78             //The root node can not be a corner.
79             //Otherwise the strength would be inf.
80             int n=0;
81             return nth_element(t, n, true);
82         }
```

7.19.3.5 std::pair<tree_element*, bool> tree_element::nth_element (int *target*, int & *n*, bool *eq_branch*) [inline, private]

Select the n'th elment of the tree.

Definition at line 232 of file faster_tree.h.

```
233         {
234             using tag::operator<<;
235             #ifndef NDEBUG
236                 if(!( (eq==0 && lt == 0 && gt == 0) || (eq!=0 && lt!=0 &&gt != 0)))
237                 {
238                     std::clog << "Error: corrupted tree\n";
239                     std::clog << tag::print << "lt" << lt;
240                     std::clog << tag::print << "eq" << eq;
241                     std::clog << tag::print << "gt" << gt;
242 
243                     abort();
244                 }
245             #endif
246 
247             if(target == n)
248                 return std::make_pair(this, eq_branch);
249             else
250             {
251                 n++;
252 
253                 tree_element * r;
254                 bool e;
255 
256                 if(eq == 0)
257                     return std::make_pair(r=0,eq_branch);
258                 else
259                 {
260                     tag::rpair(r, e) = lt->nth_element(target, n, false);
261                     if(r != NULL)
262                         return std::make_pair(r, e);
263 
264                     tag::rpair(r, e) = eq->nth_element(target, n, true);
265                     if(r != NULL)
266                         return std::make_pair(r, e);
267 
268                     return gt->nth_element(target, n, false);
269                 }
270             }
271         }
```

7.19.3.6 int tree_element::detect_corner_oriented (const CVD::Image< CVD::byte > & *im*, CVD::ImageRef *pos*, int *b*, int *n*, bool *invert*) const [inline, private]

Apply the tree to detect a corner in a single form.

Parameters:

im Image in which to detect corners

pos position at which to perform detection

b Threshold

n tree orientation to use (index in to offsets)

invert Whether to perform an intensity inversion

Returns:

0 for no corner, otherwise smallet amount by which a test passed.

Definition at line 282 of file faster_tree.h.

References detect_corner_oriented(), is_corner(), and offsets.

Referenced by detect_corner_oriented().

```
283         {
284             //Return number that threshold would have to be increased to in
285             //order to change the outcome
286 
287 
288             if(eq== NULL)
289                 return is_corner * INT_MAX;
290             else
291             {
292                 int c = im[pos];
293                 int p = im[pos + offsets[n][offset_index]];
294 
295                 const tree_element* llt = lt;
296                 const tree_element* lgt = gt;
297 
298                 if(invert)
299                     std::swap(llt, lgt);
300 
301 
302                 if(p > c+b)
303                     return std::min(p-(c+b), lgt->detect_corner_oriented(im, pos, b, n, invert));
304                 else if(p < c-b)
305                     return std::min((c-b)-p, llt->detect_corner_oriented(im, pos, b, n, invert));
306                 else
307                     return eq->detect_corner_oriented(im, pos, b, n, invert);
308             }
309         }
```

7.19.3.7 int tree_element::detect_corner (const CVD::Image< CVD::byte > & *im*, CVD::ImageRef *pos*, int *b*) const [inline]

Apply the tree in all forms to detect a corner.

Parameters:

im CVD::Image in which to detecto corners

pos position at which to perform detection

b Threshold

Returns:

0 for no corner, otherwise smallet amount by which a test passed.

Definition at line 318 of file faster_tree.h.

References invert(), and offsets.

Referenced by tree_detect_corners(), and tree_detect_corners_all().

```
319         {
```

```
320             for(int invert=0; invert <2; invert++)
321                 for(unsigned int i=0; i < offsets.size(); i++)
322                 {
323                     int n = detect_corner_oriented(im, pos, b, i, invert);
324                     if(n)
325                         return n;
326                 }
327             return 0;
328         }
```

7.19.3.8 tree_element* tree_element::copy () [inline]

Deep copy the tree.

Definition at line 331 of file faster_tree.h.

References copy(), eq, gt, and lt.

Referenced by copy(), and learn_detector().

```
332         {
333             tree_element* t = new tree_element(*this);
334             if(eq != NULL)
335             {
336                 t->lt = lt->copy();
337                 t->gt = gt->copy();
338                 t->eq = eq->copy();
339             }
340 
341             return t;
342         }
```

7.19.3.9 void tree_element::print (std::ostream & *o*, std::string *ind* = " ") const [inline]

Serialize the tree.

Parameters:

o Stream to serialize to.

ind The indent level to use for the current branch.

Definition at line 348 of file faster_tree.h.

References is_corner().

Referenced by learn_detector(), and run_learn_detector().

```
349         {
350             using tag::operator<<;
351 
352 
353             if(eq == NULL)
354                 o << ind << tag::print << "Is corner: " << is_corner << this << lt << eq << gt;
355             else
356             {
357                 o << ind << tag::print << offset_index << this << lt << eq << gt;
358                 lt->print(o, ind + "  ");
359                 eq->print(o, ind + "  ");
360                 gt->print(o, ind + "  ");
361             }
362         }
```

7.19.4 Member Data Documentation

7.19.4.1 tree_element* tree_element::lt

Branch of the tree to take if the offset pixel is much darker than the centre.

Definition at line 42 of file faster_tree.h.

Referenced by copy(), learn_detector(), make_fast_detector(), make_fast_detector_o(), and num_nodes().

7.19.4.2 tree_element* tree_element::eq

Branch of the tree to take if the offset pixel is much brighter than the centre.

Definition at line 43 of file faster_tree.h.

Referenced by copy(), is_leaf(), learn_detector(), make_fast_detector_o(), and num_nodes().

7.19.4.3 tree_element* tree_element::gt

Branch of the tree to take otherwise.

Definition at line 44 of file faster_tree.h.

Referenced by copy(), learn_detector(), make_fast_detector(), make_fast_detector_o(), and num_nodes().

7.19.4.4 bool tree_element::is_corner

If the node is a leaf, then this is its attribute.

Definition at line 45 of file faster_tree.h.

Referenced by learn_detector(), and make_fast_detector_o().

7.19.4.5 int tree_element::offset_index

Offset number of the pixel to examine. This indexes offsets[x].

Definition at line 46 of file faster_tree.h.

Referenced by learn_detector(), and make_fast_detector_o().

The documentation for this class was generated from the following file:

- faster_tree.h

Chapter 8

FAST-ER File Documentation

8.1 extract_features.cc File Reference

8.1.1 Detailed Description

Main file for the extract_features executable.

8.1.2 Usage

```
extract_features [-VAR VAL] [-exec FILE] IMAGE1 [IMAGE2 ...]
```

8.1.3 Description

This program loads a learned FAST-ER tree and extracts features so that an accelerated tree can be learned. The output is is suitable for consumption by learn_fast_tree.

The program accpets standard GVars3 commandline arguments, and the default parameters are contained in `extract_features.cfg`:

```
offsets.min_radius=2.0    //This must be the same as the value used in training
offsets.max_radius=4.2    //This must be the same as the value used in training
detector=best_faster.tree //File containing the learned FAST-ER tree
threshold=30              //Threshold at which to detect corners
skip=2                    //Skip this many pixels in the X and Y direction when extracting non-corners
debug.verify_detections=0
```

The images from which features should be extracted are specified on the commandline.

Definition in file extract_features.cc.

```
#include <gvars3/instances.h>
#include <cvd/image_io.h>
#include <stdint.h>
#include <map>
#include <iterator>
#include <string>
```

```
#include "offsets.h"
#include "faster_tree.h"
```

Functions

- void extract_feature (string &s, const BasicImage< byte > &im, const ImageRef &pos, int barrier, int o, bool invert_sense)
- int main (int argc, char **argv)

Variables

- static const char BrighterFlag = 'b'
- static const char DarkerFlag = 'd'
- static const char SimilarFlag = 's'

8.1.4 Function Documentation

8.1.4.1 void extract_feature (string & *s*, const BasicImage< byte > & *im*, const ImageRef & *pos*, int *barrier*, int *o*, bool *invert_sense*)

Extracts a feature from an image.

Parameters:

s String to extract feature in to

im Image to extract feature from

pos Location to extract feature from

barrier Threshold used to compute feature

o Index in to offsets (i.e. feature orientation) to use

invert_sense Whether or not to invert the extracted feature

Definition at line 67 of file extract_features.cc.

References BrighterFlag, DarkerFlag, num_offsets, offsets, and SimilarFlag.

Referenced by main().

```
68 {
69     int cb = im[pos] + barrier;
70     int c_b = im[pos] - barrier;
71 
72     for(int i=0; i < num_offsets; i++)
73     {
74         int pix = im[pos + offsets[o][i]];
75 
76         if(pix > cb)
77             if(invert_sense == false)
78                 s[i] = BrighterFlag;
79             else
80                 s[i] = DarkerFlag;
81         else if(pix < c_b)
82             if(invert_sense == false)
83                 s[i] = DarkerFlag;
```

```
84              else
85                  s[i] = BrighterFlag;
86          else
87              s[i] = SimilarFlag;
88      }
89  }
```

8.1.4.2 int main (int *argc*, char ** *argv*)

Driving program.

Parameters:

argc Number of commandline arguments

argv List of commandline arguments. Contains GVars3 arguments, and images to process.

Definition at line 94 of file extract_features.cc.

References create_offsets(), extract_feature(), is_corner(), load_a_tree(), num_offsets, offsets, and offsets_bbox.

```
95 {
96      //The usual initialization.
97      GUI.LoadFile("extract_features.cfg");
98      int lastarg = GUI.parseArguments(argc, argv);
99
100      create_offsets();
101
102      //Store corners and noncorners by the string representing the feature.
103      map<string, uint64_t> corners, non_corners;
104
105      //Scratch string of the correct length for extracting features in to.
106      string scratch(num_offsets, '.');
107
108      //Don't bother examining points outside this border.
109      int border = max(max(offsets_bbox.first.x, offsets_bbox.first.y), max(offsets_bbox.second.x, offse
110
111      int threshold = GV3::get<int>("threshold", 30);
112      string fname=GV3::get<string>("detector", "best_faster.tree");
113
114      //Load a detector from a tree file
115      tree_element* faster_detector;
116
117      ifstream i;
118      i.open(fname.c_str());
119
120      if(!i.good())
121      {
122          cerr << "Error: " << fname << ": " << strerror(errno) << endl;
123          exit(1);
124      }
125
126      try{
127          faster_detector = load_a_tree(i);
128      }
129      catch(ParseError p)
130      {
131          cerr << "Parse error in " << fname << endl;
132          exit(1);
133      }
134
135      //Iterate over all images, extracting features
```

```
136     for(int i=lastarg; i < argc; i++)
137     {
138         try{
139             Image<byte> im = img_load(argv[i]);
140             for(int r=border; r < im.size().y - border; r++)
141                 for(int c=border; c < im.size().x - border; c++)
142                 {
143                     ImageRef pos(c,r);
144                     //Test for cornerness
145                     bool is_corner = faster_detector->detect_corner(im, pos, threshold);
146 
147                     //Iterate over all feature orientations and inversions,
148                     //extracting the reatures, and inserting them in to the
149                     //correct bin
150                     for(unsigned int k=0; k < offsets.size(); k++)
151                         for(int l=0; l < 2; l++)
152                         {
153                             extract_feature(scratch, im, pos, threshold, k, l);
154 
155                             if(is_corner)
156                             {
157                                 corners[scratch]++;
158 
159                                 if(non_corners.count(scratch))
160                                 {
161                                     cerr << "Fatal error! extracted corner has an identical non-corner
162                                     cerr << "Are your offsets correct?\n";
163                                     exit(1);
164                                 }
165                             }
166                             else
167                             {
168                                 non_corners[scratch]++;
169                                 if(corners.count(scratch))
170                                 {
171                                     cerr << "Fatal error! extracted non-corner has an identical corner
172                                     cerr << "Are your offsets correct?\n";
173                                     exit(1);
174                                 }
175                             }
176                         }
177                 }
178             cerr << "Processed " << argv[i] << endl;
179         }
180         catch(Exceptions::All e)
181         {
182             cerr << "Failed to load " << argv[i] << ": " << e.what << endl;
183         }
184     }
185 
186     cout << num_offsets << endl;
187     copy(offsets[0].begin(), offsets[0].end(), ostream_iterator<ImageRef>(cout, " "));
188     cout << endl;
189 
190     for(map<string, uint64_t>::iterator i=corners.begin(); i != corners.end(); i++)
191         cout << i->first << " " << i->second << " 1" << endl;
192     for(map<string, uint64_t>::iterator i=non_corners.begin(); i != non_corners.end(); i++)
193         cout << i->first << " " << i->second << " 0" << endl;
194 }
```

8.1.5 Variable Documentation

8.1.5.1 const char BrighterFlag = 'b' [static]

Character code for pixels significantly brighter than the centre.

Definition at line 56 of file extract_features.cc.

Referenced by extract_feature().

8.1.5.2 const char DarkerFlag = 'd' [static]

Character code for pixels significantly darker than the centre.

Definition at line 57 of file extract_features.cc.

Referenced by extract_feature().

8.1.5.3 const char SimilarFlag = 's' [static]

Character code for pixels similar to the centre.

Definition at line 58 of file extract_features.cc.

Referenced by extract_feature().

8.2 fast_N_features.cc File Reference

8.2.1 Detailed Description

Main file for the fant_N_features executable.

8.2.2 Usage

```
./fast_N_features [-NUM N] | ./learn_fast_tree
```

8.2.3 Description

This program generates a list of all possible FAST-N features in an output format suitable for consumption by learn_fast_tree. The program accepts standarg GVars3 commandline arguments. The only useful argument is N which specifies the N for which FAST-N features should be generated.

Definition in file fast_N_features.cc.

```
#include <iostream>
#include <gvars3/instances.h>
```

Functions

- bool is_corner (const char *str, int num_for_corner, char type)
- int main (int argc, char **argv)

8.2.4 Function Documentation

8.2.4.1 bool is_corner (const char * *str*, int *num_for_corner*, char *type*) [inline]

Determine if a string has the properties of a FAST-N corner.

In other words, if it has enough consecutive characters of the correct type. This function assumes that the string wraps in a circular manner.

Parameters:

str String to test for cornerness.

num_for_corner Number of consecutive characters required for corner

type Character value which must appear consecutively#

Returns:

whether the string is a corner.

Definition at line 52 of file fast_N_features.cc.

Referenced by tree_element::detect_corner_oriented(), load_a_tree(), main(), and tree_element::print().

```
53 {
54     int num_consecutive=0;
55     int first_cons=0;
```

```
56
57     for(int i=0; i<16; i++)
58     {
59         if(str[i] == type)
60         {
61             num_consecutive++;
62
63             if(num_consecutive == num_for_corner)
64                 return 1;
65         }
66         else
67         {
68             if(num_consecutive == i)
69                 first_cons=i;
70
71             num_consecutive=0;
72         }
73     }
74
75     if(first_cons+num_consecutive >=num_for_corner)
76         return 1;
77     else
78         return 0;
79 }
```

8.2.4.2 int main (int *argc*, char ** *argv*)

This is the main function for this program.

It generates all possible FAST pixel rings using brute-force and outputs them along with their class and a uniform weighting over all features.

Parameters:

argc Number of commandline arguments

argv List of commandline arguments

Definition at line 86 of file fast_N_features.cc.

References is_corner().

```
87 {
88     GUI.parseArguments(argc, argv);
89
90     char types[]="bsd.";
91     char F[17]=".................";
92
93     int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p;
94     cout << 16 << endl;
95     cout << "[0 3] [1 3] [2 2] [3 1] [3 0] [3 -1] [2 -2] [1 -3] [0 -3] [-1 -3] [-2 -2] [-3 -1] [-3 0] [
96
97     int N = GV3::get<int>("N", 9, 1);
98
99     for(a = 0, F[ 0]='b'; a < 3; a++, F[ 0]=types[a])
100      for(b = 0, F[ 1]='b'; b < 3; b++, F[ 1]=types[b])
101       for(c = 0, F[ 2]='b'; c < 3; c++, F[ 2]=types[c])
102        for(d = 0, F[ 3]='b'; d < 3; d++, F[ 3]=types[d])
103         for(e = 0, F[ 4]='b'; e < 3; e++, F[ 4]=types[e])
104          for(f = 0, F[ 5]='b'; f < 3; f++, F[ 5]=types[f])
105           for(g = 0, F[ 6]='b'; g < 3; g++, F[ 6]=types[g])
106            for(h = 0, F[ 7]='b'; h < 3; h++, F[ 7]=types[h])
107             for(i = 0, F[ 8]='b'; i < 3; i++, F[ 8]=types[i])
108              for(j = 0, F[ 9]='b'; j < 3; j++, F[ 9]=types[j])
```

```
109               for(k = 0, F[10]='b'; k < 3; k++, F[10]=types[k])
110                for(l = 0, F[11]='b'; l < 3; l++, F[11]=types[l])
111                 for(m = 0, F[12]='b'; m < 3; m++, F[12]=types[m])
112                  for(n = 0, F[13]='b'; n < 3; n++, F[13]=types[n])
113                   for(o = 0, F[14]='b'; o < 3; o++, F[14]=types[o])
114                    for(p = 0, F[15]='b'; p < 3; p++, F[15]=types[p])
115                     cout << F << " 1 " << (is_corner(F, N, 'b') || is_corner(F, N, 'd')) << endl;
116 }
```

8.3 image_warp.cc File Reference

8.3.1 Detailed Description

Main file for the image_warp executable.

8.3.2 Usage

```
image_warp [-num NUM_IMAGES] [-dir DIR] [-type TYPE] [-out OUT_DIR]
[-stub OUT_STUB]
```

8.3.3 Description

Loads a dataset (NUM, DIR and TYPE specify the dataset according to load_data) and warp every image to look like every other image. The output is placed in the image `./dir/warp_TO_FROM.jpg`. You need to create the output directory yourself.

By flipping through the images with the same value of TO, you can see the quality of alignment within a dataset.

Definition in file image_warp.cc.

```
#include <iostream>
#include <cvd/image_io.h>
#include <cvd/image_interpolate.h>
#include <gvars3/instances.h>
#include <tag/printf.h>
#include <tag/stdpp.h>
#include "load_data.h"
#include "utility.h"
```

Functions

- Image< byte > warp_image (const Image< byte > &in, const Image< array< float, 2 > > &warp)
- int main (int argc, char **argv)

8.3.4 Function Documentation

8.3.4.1 Image<byte> warp_image (const Image< byte > & *in*, const Image< array< float, 2 > > & *warp*)

Warp one image to look like another, using bilinear interpolation.

Parameters:

in The image to warp

warp The warp to use to warp the image

Returns:

The warped image

Definition at line 60 of file image_warp.cc.

Referenced by main().

```
61 {
62     Image<byte> ret(in.size(), 0);
63 
64     image_interpolate<Interpolate::Bilinear, byte> interp(in);
65 
66     for(int y=0; y < ret.size().y; y++)
67         for(int x=0; x < ret.size().x; x++)
68         {
69             if(warp[y][x][0] != -1 && interp.in_image(Vec(warp[y][x])))
70                 ret[y][x] = interp[Vec(warp[y][x])];
71         }
72 
73     return ret;
74 }
```

8.3.4.2 int main (int *argc*, char ** *argv*)

Driving function.

Parameters:

argc Number of command line arguments

argv Commandline argument list

Definition at line 79 of file image_warp.cc.

References load_data(), and warp_image().

```
80 {
81     try
82     {
83         //Load command line arguments
84         GUI.parseArguments(argc, argv);
85 
86         vector<Image<byte> > images;
87         vector<vector<Image<array<float, 2> > > > warps;
88 
89         //Extract arguments relavent to loading a dataset
90         int n = GV3::get<int>("num", 2, 1);
91         string dir = GV3::get<string>("dir", "./", 1);
92         string format = GV3::get<string>("type", "cambridge", 1);
93 
94         //Load the dataset
95         rpair(images, warps) = load_data(dir, n, format);
96 
97         //Generate the output printf string
98         string out = GV3::get<string>("out", "./out/", 1) + "/" + GV3::get<string>("stub", "warped_%i_%
99 
100         //Warp every image to look like every other image
101         //where this makes sense.
102         for(int to = 0; to < n; to++)
103             for(int from=0; from < n; from ++)
104                 if(from != to)
```

```
105                 {
106                     Image<byte> w = warp_image(images[from], warps[to][from]);
107                     img_save(w, sPrintf(out, to, from));
108 
109                     cout << "Done " << from << " -> " << to << endl;
110                 }
111                 else
112                 {
113                     img_save(images[from], sPrintf(out, to, from));
114                 }
115     }
116     catch(Exceptions::All e)
117     {
118         cerr << "Error: " << e.what << endl;
119     }
120 }
```

8.4 learn_detector.cc File Reference

8.4.1 Detailed Description

Main file for the `learn_detector` executable.

```
learn_detector [--var value] ... [--exec config_file] ...
```

`learn_detector` reads configuration data from `learn_detector.cfg` in the current directory and acceps standard GVars3 command line arguments for setting variables and running other configuration files.

The tree is serialized by the function tree_element::print(). This tree can be extracted from the output with the following command:

`awk 'a&&!NF{exit}a;/Final tree/{a=1}'` *filename*

This file contains a direct implementation of section V of the accompanying paper, in the function learn_detector. For more information, refer to the section on optimization.

8.4.2 Configuration.

The default parameters for `learn_detector` are in `learn_detector.cfg`, which are the parameters described in to paper. They are:

```
//Training data
//Make this point to the directory containing the repeatability data
repeatability_dataset.directory=/home/edrosten/data/repeatability/box
repeatability_dataset.size=3
repeatability_dataset.format=cam

//Distince determining whether a point is repeated
fuzz=5

//Iteration parameters
Temperature.expo.scale=100
Temperature.expo.alpha=30
iterations=100000

//Threshold to use
FAST_threshold=35

//Cost function parameters
repeatability_scale=1
num_cost = 3500
max_nodes=10000

//Random tree parameters
initial_tree_depth=2
offsets.min_radius=2.0
offsets.max_radius=4.2


//Change this one for different sequences
random_seed=84175664

// Debugging
debug.print_old_tree=0
debug.print_new_tree=0
debug.verify_detections=1
debug.verify_scores=1
```

```
gvarlist
echo
echo Log starts
echo ==========
echo
```

Variables can be overridden using the `-varname value` commandline syntax. For details on how the data loading and so on operated, refer to run_learn_detector.

Definition in file learn_detector.cc.

```
#include <iostream>
#include <fstream>
#include <climits>
#include <float.h>
#include <cstring>
#include <cerrno>
#include <cmath>
#include <vector>
#include <utility>
#include <algorithm>
#include <cvd/image_io.h>
#include <cvd/random.h>
#include <cvd/vector_image_ref.h>
#include <tag/tuple.h>
#include <tag/stdpp.h>
#include <tag/fn.h>
#include <tag/printf.h>
#include <TooN/TooN.h>
#include "gvars_vector.h"
#include "faster_tree.h"
#include "faster_bytecode.h"
#include "offsets.h"
#include "utility.h"
#include "load_data.h"
```

Functions

- double sq (double d)
- vector< int > range (int num)
- vector< ImageRef > generate_disc (int radius)
- Image< bool > paint_circles (const vector< ImageRef > &corners, const vector< ImageRef > &circle, ImageRef size)

- float compute_repeatability (const vector< vector< Image< array< float, 2 > > > > &warps, const vector< vector< ImageRef > > &corners, int r, ImageRef size)
- tree_element * random_tree (int d, bool is_eq_branch=1)
- double compute_temperature (int i, int imax)
- tree_element * learn_detector (const vector< Image< byte > > &images, const vector< vector< Image< array< float, 2 > > > > &warps)
- void run_learn_detector (int argc, char **argv)
- int main (int argc, char **argv)

8.5 learn_fast_tree.cc File Reference

8.5.1 Detailed Description

Main file for the `learn_fast_tree` executable.

`learn_fast_tree [-weight.`*x weight*`] ... < `*infile* `>` *outfile*

`learn_fast_tree` used ID3 to learn a ternary decision tree for corner detection. The data is read from the standard input, and the tree is written to the standard output. This is designed to learn FAST feature detectors, and does not allow for the possibility ambbiguity in the input data.

8.5.2 Input data

The input data has the following format:

```
5
[-1 -1] [1 1] [3 4] [5 6] [-3 4]
bbbbb 1     0
bsdsb 1000  1
.
.
.
```

The first row is the number of features. The second row is the the list of offsets assosciated with each feature. This list has no effect on the learning of the tree, but it is passed through to the outpur for convinience.

The remaining rows contain the data. The first field is the ternary feature vector. The three characters "b", "d" and "s" are the correspond to brighter, darker and similar respectively, with the first feature being stored in the first character and so on.

The next field is the number of instances of the particular feature. The third field is the class, with 1 for corner, and 0 for background.

8.5.2.1 Generating input data

Ideally, input data will be generated from some sample images. The program FIXME can be used to do this.

Additionally, a the program fast_N_features can be used to generate all possible feature combinations for FAST-N features. When run without arguments, it generates data for FAST-9 features, otherwise the argument can be used to specify N.

8.5.3 Output data

The program does not generate source code directly, rather it generates an easily parsabel representation of a decision tree which can be turned in to source code.

The structure of the tree is described in detail in print_tree.

Definition in file learn_fast_tree.cc.

Classes

- struct datapoint< FEATURE_SIZE >

This structure represents a datapoint.

- struct tree

  This class represents a decision tree.

Defines

- #define fatal(E, S,...) vfatal((E), (S), (tag::Fmt,## __VA_ARGS__))

Enumerations

- enum Ternary { Brighter = 'b', Darker = 'd', Similar = 's' }

Functions

- template<class C>
  void vfatal (int err, const string &s, const C &list)
- template<int S>
  V_tuple< shared_ptr< vector< datapoint< S > > >, uint64_t >::type load_features (unsigned int nfeats)
- double entropy (uint64_t n, uint64_t c1)
- template<int S>
  int find_best_split (const vector< datapoint< S > > &fs, const vector< double > &weights, unsigned int nfeats)
- template<int S>
  shared_ptr< tree > build_tree (vector< datapoint< S > > &corners, const vector< double > &weights, int nfeats)
- void print_tree (const tree *node, ostream &o, const string &i="")
- template<int S>
  V_tuple< shared_ptr< tree >, uint64_t >::type load_and_build_tree (unsigned int num_features, const vector< double > &weights)
- int main (int argc, char **argv)

8.5.4 Enumeration Type Documentation

8.5.4.1 enum Ternary

Representations of ternary digits.

Enumerator:

Brighter

Darker

Similar

Definition at line 114 of file learn_fast_tree.cc.

```
115 {
116     Brighter='b',
117     Darker  ='d',
118     Similar ='s'
119 };
```

8.5.5 Function Documentation

8.5.5.1 template<int S> V_tuple<shared_ptr<vector<datapoint<S> > >, uint64_t >::type load_features (unsigned int *nfeats*) [inline]

This function loads as many datapoints from the standard input as possible.

Datapoints consist of a feature vector (a string containing the characters "b", "d" and "s"), a number of instances and a class.

See datapoint::pack_trits for a more complete description of the feature vector.

The tokens are whitespace separated.

Parameters:

nfeats Number of features in a feature vector. Used to spot errors.

Returns:

Loaded datapoints and total number of instances.

Definition at line 246 of file learn_fast_tree.cc.

References fatal.

```
247 {
248     shared_ptr<vector<datapoint<S> > > ret(new vector<datapoint<S> >);
249
250
251     string unpacked_feature;
252
253     uint64_t total_num = 0;
254
255     uint64_t line_num=2;
256
257     for(;;)
258     {
259         uint64_t count;
260         bool is;
261
262         cin >> unpacked_feature >> count >> is;
263
264         if(!cin)
265             break;
266
267         line_num++;
268
269         if(unpacked_feature.size() != nfeats)
270             fatal(1, "Feature string length is %i, not %i on line %i", unpacked_feature.size(), nfeats
271
272         if(count == 0)
273             fatal(4, "Zero count is invalid");
274
275         ret->push_back(datapoint<S>(unpacked_feature, count, is));
276
```

```
277         total_num += count;
278     }
279
280     cerr << "Num features: " << total_num << endl
281          << "Num distinct: " << ret->size() << endl;
282
283     return make_vtuple(ret, total_num);
284 }
```

8.5.5.2 double entropy (uint64_t *n*, uint64_t *c1*)

Compute the entropy of a set with binary annotations.

Parameters:

n Number of elements in the set

c1 Number of elements in class 1

Returns:

The set entropy.

Definition at line 291 of file learn_fast_tree.cc.

Referenced by find_best_split().

```
292 {
293     assert(c1 <= n);
294     //n is total number, c1 in num in class 1
295     if(n == 0)
296         return 0;
297     else if(c1 == 0 || c1 == n)
298         return 0;
299     else
300     {
301         double p1 = (double)c1 / n;
302         double p2 = 1-p1;
303
304         return -(double)n*(p1*log(p1) + p2*log(p2)) / log(2.f);
305     }
306 }
```

8.5.5.3 template<int S> int find_best_split (const vector< datapoint< S > > & *fs*, const vector< double > & *weights*, unsigned int *nfeats*) [inline]

Find the feature that has the highest weighted entropy change.

Parameters:

fs datapoints to split in to three subsets.

weights weights on features

nfeats Number of features in use.

Returns:

best feature.

Definition at line 313 of file learn_fast_tree.cc.

References Brighter, Darker, entropy(), fatal, and Similar.

```
314 {
315     assert(nfeats == weights.size());
316     uint64_t num_total = 0, num_corners=0;
317 
318     for(typename vector<datapoint<S> >::const_iterator i=fs.begin(); i != fs.end(); i++)
319     {
320         num_total += i->count;
321         if(i->is_a_corner)
322             num_corners += i->count;
323     }
324 
325     double total_entropy = entropy(num_total, num_corners);
326 
327     double biggest_delta = 0;
328     int   feature_num = -1;
329 
330     for(unsigned int i=0; i < nfeats; i++)
331     {
332         uint64_t num_bri = 0, num_dar = 0, num_sim = 0;
333         uint64_t cor_bri = 0, cor_dar = 0, cor_sim = 0;
334 
335         for(typename vector<datapoint<S> >::const_iterator f=fs.begin(); f != fs.end(); f++)
336         {
337             switch(f->get_trit(i))
338             {
339                 case Brighter:
340                     num_bri += f->count;
341                     if(f->is_a_corner)
342                         cor_bri += f->count;
343                     break;
344 
345                 case Darker:
346                     num_dar += f->count;
347                     if(f->is_a_corner)
348                         cor_dar += f->count;
349                     break;
350 
351                 case Similar:
352                     num_sim += f->count;
353                     if(f->is_a_corner)
354                         cor_sim += f->count;
355                     break;
356             }
357         }
358 
359         double delta_e = total_entropy - (entropy(num_bri, cor_bri) + entropy(num_dar, cor_dar) + entr
360 
361         delta_e *= weights[i];
362 
363         if(delta_e > biggest_delta)
364         {
365             biggest_delta = delta_e;
366             feature_num = i;
367         }
368     }
369 
370     if(feature_num == -1)
371         fatal(3, "Couldn't find a split.");
372 
373     return feature_num;
374 }
```

8.5.5.4 template<int S> shared_ptr<tree> build_tree (vector< datapoint< S > > & *corners*, const vector< double > & *weights*, int *nfeats*) [inline]

This function uses ID3 to construct a decision tree.

The entropy changes are weighted by the list of weights, to allow bias towards certain features. This function assumes that the class is an exact function of the data. If there datapoints with different classes share the same feature vector, the program will crash with error code 3.

Parameters:

corners Datapoints in this part of the subtree to classify

weights Weights on the features

nfeats Number of features actually used

Returns:

The tree required to classify corners

Definition at line 468 of file learn_fast_tree.cc.

References Brighter, tree::CornerLeaf(), Darker, tree::NonCornerLeaf(), and Similar.

```
469 {
470     //Find the split
471     int f = find_best_split<S>(corners, weights, nfeats);
472 
473     //Split corners in to the three chunks, based on the result of find_best_split.
474     //Also, count how many of each class ends up in each of the three bins.
475     //It may apper to be inefficient to use a vector here instead of a list, in terms
476     //of memory, but the per-element storage overhead of the list is such that it uses
477     //considerably more memory and is much slower.
478     vector<datapoint<S> > brighter, darker, similar;
479     uint64_t num_bri=0, cor_bri=0, num_dar=0, cor_dar=0, num_sim=0, cor_sim=0;
480 
481     for(size_t i=0; i < corners.size(); i++)
482     {
483         switch(corners[i].get_trit(f))
484         {
485             case Brighter:
486                 brighter.push_back(corners[i]);
487                 num_bri += corners[i].count;
488                 if(corners[i].is_a_corner)
489                     cor_bri += corners[i].count;
490                 break;
491 
492             case Darker:
493                 darker.push_back(corners[i]);
494                 num_dar += corners[i].count;
495                 if(corners[i].is_a_corner)
496                     cor_dar += corners[i].count;
497                 break;
498 
499             case Similar:
500                 similar.push_back(corners[i]);
501                 num_sim += corners[i].count;
502                 if(corners[i].is_a_corner)
503                     cor_sim += corners[i].count;
504                 break;
505         }
506     }
507 
508     //Deallocate the memory now it's no longer needed.
```

```
509     corners.clear();
510
511     //This is not the same as corners.size(), since the corners (datapoints)
512     //have a count assosciated with them.
513     uint64_t num_tests =  num_bri + num_dar + num_sim;
514
515
516     //Build the subtrees
517     shared_ptr<tree> b_tree, d_tree, s_tree;
518
519
520     //If the sublist contains a single class, then instantiate a leaf,
521     //otherwise recursively build the tree.
522     if(cor_bri == 0)
523         b_tree = tree::NonCornerLeaf(num_bri);
524     else if(cor_bri == num_bri)
525         b_tree = tree::CornerLeaf(num_bri);
526     else
527         b_tree = build_tree<S>(brighter, weights, nfeats);
528
529
530     if(cor_dar == 0)
531         d_tree = tree::NonCornerLeaf(num_dar);
532     else if(cor_dar == num_dar)
533         d_tree = tree::CornerLeaf(num_dar);
534     else
535         d_tree = build_tree<S>(darker, weights, nfeats);
536
537
538     if(cor_sim == 0)
539         s_tree = tree::NonCornerLeaf(num_sim);
540     else if(cor_sim == num_sim)
541         s_tree = tree::CornerLeaf(num_sim);
542     else
543         s_tree = build_tree<S>(similar, weights, nfeats);
544
545     return shared_ptr<tree>(new tree(b_tree, d_tree, s_tree, f, num_tests));
546 }
```

8.5.5.5 void print_tree (const tree * *node*, ostream & *o*, const string & *i* = " ")

This function traverses the tree and produces a textual representation of it.

Additionally, if any of the subtrees are the same, then a single subtree is produced and the test is removed.

A subtree has the following format:

```
subtree= lead | node;

leaf = "corner" | "background" ;

node = node2 | node3;

node3 = "if_brighter" feature_number n1 n2 n3
            subtree
        "elsf_darker" feature_number
            subtree
        "else"
            subtree
        "end";

 node2= if_statement feature_number n1 n2
            subtree
        "else"
            subtree
```

```
        "end";

if_statement = "if_brighter" | "if_darker" | "if_either";
feature_number ==integer;
n1 = integer;
n2 = integer;
n3 = integer;
```

feature_number refers to the index of the feature that the test is performed on.

In *node3*, a 3 way test is performed. *n1*, *n2* and *n3* refer to the number of training examples landing in the *if* block, the *elfs* block and the *else* block respectivly.

In a *node2* node, one of the tests has been removed. *n1* and *n2refer* to the number of training examples landing in the *if* block and the *else* block respectivly.

Although not mentioned in the grammar, the indenting is kept very strict.

This representation has been designed to be parsed very easily with simple regular expressions, hence the use if "elsf" as opposed to "elif" or "elseif".

Parameters:

node (sub)tree to serialize

o Stream to serialize to.

i Indent to print before each line of the serialized tree.

Definition at line 601 of file learn_fast_tree.cc.

References tree::brighter, tree::Corner, tree::darker, tree::feature_to_test, tree::is_a_corner, tree::NonCorner, tree::num_datapoints, and tree::similar.

Referenced by main().

```
602 {
603     if(node->is_a_corner == tree::Corner)
604         o << i << "corner" << endl;
605     else if(node->is_a_corner == tree::NonCorner)
606         o << i << "background" << endl;
607     else
608     {
609         string b = node->brighter->stringify();
610         string d = node->darker->stringify();
611         string s = node->similar->stringify();
612 
613         const tree * bt = node->brighter.get();
614         const tree * dt = node->darker.get();
615         const tree * st = node->similar.get();
616         string ii = i + " ";
617 
618         int f = node->feature_to_test;
619 
620         if(b == d && d == s) //All the same
621         {
622             //o << i << "if " << f << " is whatever\n";
623             print_tree(st, o, i);
624         }
625         else if(d == s)  //Bright is different
626         {
627             o << i << "if_brighter " << f << " " << bt->num_datapoints << " " << dt->num_datapoints+st
628                 print_tree(bt, o, ii);
629             o << i << "else" << endl;
630                 print_tree(st, o, ii);
```

```
631             o << i << "end" << endl;
632
633         }
634         else if(b == s) //Dark is different
635         {
636             o << i << "if_darker " << f << " " << dt->num_datapoints << " " << bt->num_datapoints + st
637                 print_tree(dt, o, ii);
638             o << i << "else" << endl;
639                 print_tree(st, o, ii);
640             o << i << "end" << endl;
641         }
642         else if(b == d) //Similar is different
643         {
644             o << i << "if_either " << f << " " <<  bt->num_datapoints + dt->num_datapoints  << " " <<
645                 print_tree(bt, o, ii);
646             o << i << "else" << endl;
647                 print_tree(st, o, ii);
648             o << i << "end" << endl;
649         }
650         else //All different
651         {
652             o << i << "if_brighter " << f << " "  <<  bt->num_datapoints << " " << dt->num_datapoints
653                 print_tree(bt, o, ii);
654             o << i << "elsf_darker " << f << endl;
655                 print_tree(dt, o, ii);
656             o << i << "else" << endl;
657                 print_tree(st, o, ii);
658             o << i << "end" << endl;
659         }
660     }
661 }
```

8.5.5.6 template<int S> V_tuple<shared_ptr<tree>, uint64_t>::type load_and_build_tree (unsigned int *num_features*, const vector< double > & *weights*) [inline]

This function loads data and builds a tree.

It is templated because datapoint is templated, for reasons of memory efficiency.

Parameters:

num_features Number of features used

weights Weights on each feature.

Returns:

The learned tree, and number of datapoints.

Definition at line 668 of file learn_fast_tree.cc.

```
669 {
670     assert(weights.size() == num_features);
671
672     shared_ptr<vector<datapoint<S> > > l;
673     uint64_t num_datapoints;
674
675     //Load the data
676     make_rtuple(l, num_datapoints) = load_features<S>(num_features);
677
678     cerr << "Loaded.\n";
679
680     //Build the tree
```

```
681     shared_ptr<tree> tree;
682     tree  = build_tree<S>(*l, weights, num_features);
683
684     return make_vtuple(tree, num_datapoints);
685 }
```

8.5.5.7 int main (int *argc*, char ** *argv*)

The main program.

Parameters:

argc Number of commandline arguments

argv Commandline arguments

Each feature takes up 2 bits. Since GCC doesn't pack any finer then 32 bits for hetrogenous structs, there is no point in having granularity finer than 16 features.

Definition at line 692 of file learn_fast_tree.cc.

References fatal, offsets, and print_tree().

```
693 {
694     //Set up default arguments
695     GUI.parseArguments(argc, argv);
696
697     cin.sync_with_stdio(false);
698     cout.sync_with_stdio(false);
699
700
701     ////////////////////
702     //read file
703
704     //Read number of features
705     unsigned int num_features;
706     cin >> num_features;
707     if(!cin.good() || cin.eof())
708         fatal(6, "Error reading number of features.");
709
710     //Read offset list
711     vector<ImageRef> offsets(num_features);
712     for(unsigned int i=0; i < num_features; i++)
713         cin >> offsets[i];
714     if(!cin.good() || cin.eof())
715         fatal(7, "Error reading offset list.");
716
717     //Read weights for the various offsets
718     vector<double> weights(offsets.size());
719     for(unsigned int i=0; i < weights.size(); i++)
720         weights[i] = GV3::get<double>(sPrintf("weights.%i", i), 1, 1);
721
722
723     shared_ptr<tree> tree;
724     uint64_t num_datapoints;
725
726     ///Each feature takes up 2 bits. Since GCC doesn't pack any finer
727     ///then 32 bits for hetrogenous structs, there is no point in having
728     ///granularity finer than 16 features.
729     if(num_features <= 16)
730         make_rtuple(tree, num_datapoints) = load_and_build_tree<16>(num_features, weights);
731     else if(num_features <= 32)
732         make_rtuple(tree, num_datapoints) = load_and_build_tree<32>(num_features, weights);
```

```
733     else if(num_features <= 48)
734         make_rtuple(tree, num_datapoints) = load_and_build_tree<48>(num_features, weights);
735     else if(num_features <= 64)
736         make_rtuple(tree, num_datapoints) = load_and_build_tree<64>(num_features, weights);
737     else
738         fatal(8, "Too many feratures (%i). To learn from this, see %s, line %i.", num_features, __FILE
739
740
741     cout << num_features << endl;
742     copy(offsets.begin(), offsets.end(), ostream_iterator<ImageRef>(cout, " "));
743     cout << endl;
744     print_tree(tree.get(), cout);
745 }
```

8.6 test_repeatability.cc File Reference

8.6.1 Detailed Description

Main file for the test_repeatability executable.

8.6.2 Usage

```
test_repeatability [-var VAL] [-exec FILE]
```

8.6.3 Description

This program loads a dataset and then computes repeatability of the specified detector on the dataset. This program accepts standard GVars3 commandline arguments and loads `learn_detector.cfg` as a the default configuration:

```
////////////////////////////////////////////////////////////////////////////
//
// Parameters for various corner detectors
//

//Parameters for the Harris and Shi-Tomasi detector
harris.blur=2.5        //Standard deviation of blur
harris.sigmas=2.0      //Blur to this many standard deviations

shitomasi.blur=2.5
shitomasi.sigmas=2.0

//Parameters for DoG detector
dog.sigma=1.0                //Initial blur
dog.divisions_per_octave=3
dog.octaves=4

//Paramaters for Harris-Laplace detector
harrislaplace.harris.blur=1
harrislaplace.harris.sigmas=2
harrislaplace.dog.sigma=1
harrislaplace.dog.divisions_per_octave=7
harrislaplace.dog.octaves=4

//SUSAN detector
susan.dt=4

///Parameters for FAST-ER tree detector.
offsets.min_radius=2.0  //This must be the same as the value used in training
offsets.max_radius=4.2  //This must be the same as the value used in training

debug.verify_detections=0 //Debugging code. Leave to 0.
debug.verify_scores=0

faster2=best_faster.tree  //Detector file to load

////////////////////////////////////////////////////////////////////////////
//
// Parameters for the experiment
//

// Number of corners per frame to use
cpf=0 10 20 30 40 50 60 70 80 90 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950 1
```

```
ncpf=500      //Number of corners per frame to use for the noise test
nmax=50       //Noise standard deviation to run to in the noise test
r=5           //Radius used to determine if the point is repeated
test="normal" //Type of test to run. Options are normal or noise
```

The available detectors are selected using the `detector` variable. Options are given in get_detector.

Definition in file test_repeatability.cc.

```
#include <iostream>
#include <sstream>
#include <cfloat>
#include <map>
#include <utility>
#include <cvd/image_io.h>
#include <cvd/random.h>
#include <cvd/image_interpolate.h>
#include <tag/printf.h>
#include <tag/stdpp.h>
#include "gvars_vector.h"
#include "load_data.h"
#include "detectors.h"
#include "utility.h"
```

Functions

- double compute_repeatability_exact (const vector< vector< Image< array< float, 2 > > > > &warps, const vector< vector< ImageRef > > &corners, double r)
- void compute_repeatability_all (const vector< Image< byte > > &images, const vector< vector< Image< array< float, 2 > > > > &warps, const DetectN &detector, const vector< int > &cpf, double fuzz)
- void compute_repeatability_noise (const vector< Image< byte > > &images, const vector< vector< Image< array< float, 2 > > > > &warps, const DetectN &detector, int cpf, float n, double fuzz)
- void mmain (int argc, char **argv)
- int main (int argc, char **argv)

8.6.4 Function Documentation

8.6.4.1 int main (int *argc*, char ** *argv*)

Driving function which catches exceptions.

Parameters:

argc Number of command line arguments

argv Commandline argument list

Definition at line 231 of file test_repeatability.cc.

References mmain().

```
232 {
233     try
234     {
235         mmain(argc, argv);
236     }
237     catch(Exceptions::All e)
238     {
239         cerr << "Error: " << e.what << endl;
240     }
241 }
```

8.7 warp_to_png.cc File Reference

8.7.1 Detailed Description

Main file for the warp_to_png executable.

8.7.2 Usage

`warp_to_png [-size "x y"] <` *infile.warp* `>` *outfile.png*

8.7.3 Description

Converts a text warp file in to a PNG warp file. The size is used to specify the image shape to convert to and defaults to 768 by 576.

Definition in file warp_to_png.cc.

```
#include <iostream>
#include <iterator>
#include <vector>
#include <cstdlib>
#include <cvd/image_io.h>
#include <gvars3/instances.h>
#include "warp_to_png.h"
```

Functions

- int main (int argc, char **argv)

8.7.4 Function Documentation

8.7.4.1 int main (int *argc*, char ** *argv*)

Driving function.

Parameters:

argc Number of command line arguments

argv Commandline argument list

Definition at line 57 of file warp_to_png.cc.

```
58 {
59     try
60     {
61         GUI.parseArguments(argc, argv);
62 
63         ImageRef size = GV3::get<ImageRef>("size", ImageRef(768,576), 1);
64 
```

```
65
66          Image<Rgb<unsigned short> > si(size);
67
68          for(int y=0; y < size.y; y++)
69              for(int x=0; x < size.x; x++)
70              {
71                  float f1, f2;
72                  cin >> f1 >> f2;
73
74                  if(!cin.good())
75                  {
76                      cerr << "EOF!\n";
77                      exit(1);
78                  }
79
80                  if(f1 < -5 || f1 > 1000)
81                  {
82                      cerr << "Bad value at " << x << ", " << y << ": " << f1;
83                      exit(2);
84                  }
85
86                  if(f2 < -5 || f2 > 1000)
87                  {
88                      cerr << "Bad value at " << x << ", " << y << ": " << f2;
89                      exit(2);
90                  }
91
92                  Rgb<unsigned short> o;
93
94                  o.red = (unsigned short) ((SHIFT + f1)*MULTIPLIER + .5);
95                  o.green = (unsigned short) ((SHIFT + f2)*MULTIPLIER + .5);
96                  o.blue = 0;
97
98                  si[y][x] = o;
99              }
100
101         img_save(si, cout, ImageType::PNG);
102     }
103     catch(Exceptions::All e)
104     {
105         cerr << "Error: " << e.what << endl;
106         return 1;
107     }
108 }
```

Index