API for C Implementation of Blob Detection Algorithm

Per-Erik Forssén

Computer Vision Laboratory, Department of Electrical Engineering

Linköping University, SE-581 83 Linköping, Sweden

April 16, 2004

1 Introduction

This document is a documentation of the C implementation of a blob detection algorithm described in chapter 7 of [1].

All source code files in the package are listed in section 2. For all `.c`-source files (except the application program) there is also a corresponding `.h` file, which should be `include`-d by the application using the methods in this file.

The implementation contains the example application `blobdemo_ppm`, which reads an RGB image in PPM format, and writes a new PPM file with the found blobs painted on a green background.

The perhaps most interesting function is the function `extract_blobs` contained in file `extract_blobs.c`. Its arguments are described in table 7. Basically it takes a pointer to an image, and some algorithm parameters as input, and outputs detected blobs in three lists of blob properties.

If you find this document too brief, please have a look at the example program `blobdemo_ppm.c` in section 3, for an example of how to use the API.

2 List of Files

- **`blobdemo_ppm.c`**
  This is the application program. It is listed in section 3 .
  Usage: `blobdemo_ppm <image.ppm> <blobs.ppm> [<regions.ppm>] [<dmax>]`
  The program reads an RGB image in the PPM format and outputs a new PPM-file with the found blobs painted on a green background. The optional out argument `regions.ppm` is a visualisation of the regions used to compute the blobs. The optional parameter `dmax` controls the colour sensitivity. Default is `dmax=0.16`.

- **`extract_blobs.c`**
  This file contains methods that encapsulate much of the details in blob extraction. See tables 6 and 7 for a list of methods.

- **file_and_time.c**
  This file contains routines for managing file IO, and execution timers. See table 8 for a list of methods.

- **image_buffer.c**
  Methods for `buffer` and `ibuffer` data types. A `buffer` or an `ibuffer` is a container for a 3D array, typically an image.
  For easy switching between `double` and `float` precision of floating point numbers, the data type `fpnum` is declared as either `double` or `float` in `image_buffer.h`. On a 64-bit architecture, the `double` option is actually faster.
  Fields and methods of the `buffer` data type are listed in tables 1 and 2.
  Fields and methods of the `ibuffer` data type are listed in tables 3 and 4.

- **merge_blobs.c**
  Methods to merge and clean up a list of blobs. See table 9.

- **pnmio.c**
  This file contains a set of functions for reading and writing `pnm` file headers. Table 5 lists the methods. The actual data in the file should be read or written using an `fread` call. See the file `file_and_time.c` for an example of how to use the methods in `pnmio.c`.

- **region_image.c**
  Methods to build a region image, and to compute moments from the region image. See table 10.

- **sbinfilt.c**
  This file implements the non-linear filter that is used to build the clustering pyramid. The methods are listed in table 11. See [1] for details of the algorithm.

- **visualise.c**
  This file contains routines for blob visualisation. I.e. the code that generates an image with ellipses representing the blobs. See table 12 for a list of methods.

Field	Type	Description
`rows`	`int`	Number of rows in array
`cols`	`int`	Number of columns in array
`ndim`	`int`	Number of fields in array, e.g. 3 for an RGB image
`data`	`fpnum`	Pointer to floating point data.

Table 1: The `buffer` data type

Method	Return type	Argument description
buffer_new	buffer	int rows, int cols, int ndims: These define the size of the buffer to allocate.
buffer_pdims		buffer bf: The buffer to print dimension info of to stdout.
buffer_free		buffer bf: A buffer to be released.

Table 2: Methods for the buffer data type

Field	Type	Description
rows	int	Number of rows in array
cols	int	Number of columns in array
ndim	int	Number of fields in array, e.g. 3 for an RGB image
data	int	Pointer to integer data.

Table 3: The ibuffer data type

Method	Return type	Argument description
ibuffer_new	buffer	int rows, int cols, int ndims: These define the size of the ibuffer to allocate.
ibuffer_pdims		ibuffer bf: The ibuffer to print dimension info of to stdout.
ibuffer_free		ibuffer bf: An ibuffer to be released.

Table 4: Methods for the ibuffer data type

Method	Return type	Argument list
pnm_readhead		char *name Filename int *format Location to store image type tag int *height Location to store image height int *width Location to store image width
pnm_writehead	FILE *	char *name Filename int format Tag for desired format int height Height of image int width Width of image
pnm_close		FILE *f Handle of file to close

Table 5: Methods in the pnmio.c file

Method	Return type	Argument list
number_of_scales	int	Calculate number of scales required to build an octave pyramid of an image. buffer *bf_image Input image
imk_pyramid_new	buffer **	Create a pyramid and insert input image at scale 0. buffer *bf_image Input image
imc_pyramid_new	ibuffer **	Create a certainty pyramid and insert input certainty at scale 0. ibuffer *bf_imc Input certainty image
set_to_ones		Set an ibuffer to all ones. The input is assumed to be of size $M \times N \times 1$. ibuffer *bf_image The input array.
sbinfilt_pyramid		Generate a clustering pyramid by successive filtering of bl_imk and bl_imc. buffer **bl_imk Image pyramid ibuffer **bl_imc Certainty pyramid int nsc Number of scales fpnum dmax Maximum allowed property distance fpnum cmin Weighted fraction of pixels reqired for $c = 1$. int roi_side $2 \rightarrow 2 \times 2$, $4 \rightarrow 4 \times 4$, $6 \rightarrow 6 \times 6 \ldots$ int miter Number of M-estimation steps to follow.
make_label_image		Generate a label image from a clustering pyramid. buffer **bl_result List of 4 result arrays buffer **bl_imk Input image pyramid ibuffer **bl_imc Input certainty pyramid int nsc Number of scales int lowsc Scale to stop assigning new labels at fpnum dmax Maximum allowed property distance
merge_and_cleanup		Merge blobs and clean up blob list. buffer **bl_result List of 4 result arrays buffer *bf_mvec1 Moment vector list buffer *bf_pvec1 Property vector list ibuffer *bf_csc1 Detection scales ibuffer *bf_cntl Overlap count list fpnum minc Merger threshold int amin Minimum required area fpnum dmax Maximum allowed property distance

Table 6: Methods in extract_blobs.c part 1.

Method	Return type	Argument list
`extract_blobs`		Encapsulated blob feature extraction algorithm. `buffer *bf_image` Input image `ibuffer *bf_cert` Input certainty `buffer **bl_lout` List of 4 result arrays `fpnum dmax` Maximum allowed property distance `fpnum cmin` Weighted fraction of pixels reqired for $c = 1$. `int roi_side` $2 \rightarrow 2 \times 2$, $4 \rightarrow 4 \times 4$, $6 \rightarrow 6 \times 6 \ldots$ `int miter` Number of M-estimation steps to follow. `int lowsc` Scale to stop assigning new labels at `fpnum minc` Merger threshold `int amin` Minimum required area

Table 7: Methods in `extract_blobs.c` part 2.

Method	Return type	Argument list
`write_time_diff`		Write difference between two `clock_t` structs to standard output. `char *strg` Message preceeding time text `clock_t t0` Start time `clock_t t1` End time
`read_pnm_file`		Read a file from disk using PNMIO. See also file `pnmio.c` `char *fname` Name of file to read `char *pname` Name of program (for error message) `buffer **bf_image` Place to store resultant image buffer `int ssfl` Subsample image if non-zero
`dump_to_file`		Dump a `buffer` to file in ascii form suitable to be read as an `.m` file in Matlab. `FILE *out_fid` open file stream `char *vname` string containing variable name `buffer *bf_var` array holding data
`idump_to_file`		Dump an `ibuffer` to file in ascii form suitable to be read as an `.m` file in Matlab. `FILE *out_fid` open file stream `char *vname` string containing variable name `ibuffer *bf_var` array holding data

Table 8: Methods in `file_and_time.c`

Method	Return type	Argument list
`merge_blobs`		This method merges two blobs `fpnum *mvec1` Moment vector for blob 1 `fpnum *mvec2` Moment vector for blob 2 `fpnum *mvecn` Output moment vector `fpnum *pvec1` Property vector for blob 1 `fpnum *pvec2` Property vector for blob 2 `fpnum *pvecn` Output property vector `int ndim` Number of property dimensions
`bloblist_merge_cnt`	`int`	Old merge function. Returns number of merged regions `buffer *bf_mvec0` Input moment vectors `buffer *bf_pvec0` Input property vectors `buffer *bf_mvecn` Output moment vectors `buffer *bf_pvecn` Output property vectors `ibuffer *bf_out_ind` Index pointer list `ibuffer *bf_cntl` Overlap count list `fpnum minc` Merger threshold
`bloblist_merge_cnt2`	`int`	New merge function. More expensive, but better. Returns number of merged regions `buffer *bf_mvec0` Input moment vectors `buffer *bf_pvec0` Input property vectors `buffer *bf_mvecn` Output moment vectors `buffer *bf_pvecn` Output property vectors `ibuffer *bf_out_ind` Index pointer list `ibuffer *bf_cntl` Overlap count list `fpnum minc` Merger threshold `fpnum dmax2` Squared max property distance
`bloblist_mark_invalid`	`int`	Discard blobs with $\det \boldsymbol{I} \leq 0$ or $a < a_{\min}$, by setting their area to zero, and `out_ind[k]=0` `buffer *bf_mvec` Moment vectors `ibuffer *bf_out_ind` Array of index pointers `int amin` Minimum required area
`bloblist_compact`		Remove holes in bloblists after `bloblist_merge_cnt` `buffer *bf_mvecn` Input moment vectors `buffer *bf_pvecn` Input property vectors `ibuffer *bf_cscn` Input detection scales `buffer *bf_mvecm` Output moment vectors `buffer *bf_pvecm` Output property vectors `ibuffer *bf_cscm` Output detection scales `ibuffer *bf_out_ind` Index pointer list

Table 9: Methods in `merge_blobs.c`

Method	Return type	Argument list
propagate_regions	int	ibuffer bf_labelim1 Input label image $(Y \times X)$ ibuffer bf_labelim2 Output label image $(Y \times X)$ buffer bf_imk Property image $(Y \times X \times D)$ ibuffer bf_imc Confidence image $(Y \times X)$ buffer bf_pvec Prototype list $(D \times N)$ fpnum dmax2 Squared max property distance Returns number of new seeds (for later allocation by find_new_seeds)
find_new_seeds		ibuffer *bf_labelim1 Input label image $(Y \times X)$ buffer *bf_imk Property image $(Y \times X \times D)$ ibuffer *bf_imc Confidence image $(Y \times X)$ buffer *bf_pvec1 Input prototype list $(D \times N)$ buffer *bf_pvec2 Output prototype list $(D \times N_{\text{new}})$ int regions Length of pvec1 int new_seeds Number of new seeds (as found by propagate_regions)
propagate_regions_cnt		ibuffer *bf_labelim1 Input label image $(Y/2 \times X/2)$ ibuffer *bf_labelim2 Output label image $(Y \times X)$ buffer *bf_imk Property image $(Y \times X \times D)$ ibuffer *bf_imc Confidence image $(Y \times X)$ buffer *bf_pvec Prototype list $(D \times N)$ ibuffer *bf_cntl Boundary count list fpnum dmax2 Squared max property distance
compute_moments		ibuffer *bf_labelim Label image $(Y \times X)$ buffer *bf_image RGB image $(Y \times X \times D)$ buffer *bf_pvec Output property averages buffer *bf_mvec Output moments
labelim_compact		Loop over label image and replace old labels with new that are compatible with bf_mvec and bf_pvec lists. ibuffer *bf_labelim Label image to modify $(Y \times X)$ ibuffer *bf_out_ind Compaction list $(1 \times N)$

Table 10: Methods in region_image.c

Method	Return type	Argumentlist
binfilt2d	int *	int order Allocates space and returns an array containing an outer product of two binomial filters of given order.
sbinfilt2d		buffer *bf_im0 Input image buffer ibuffer *bf_ic0 Input confidence map buffer *bf_im1 Location of result image ibuffer *bf_ic1 Location of result confidence fpnum dmax2 Squared max property (colour) distance fpnum cmin Weighted fraction of pixels reqired for $c = 1$ int roi_side $2 \rightarrow 2 \times 2$, $4 \rightarrow 4 \times 4$, $6 \rightarrow 6 \times 6 \ldots$ int miter Number of M-estimation steps to follow.

Table 11: Methods in sbinfilt.c

Method	Return type	Argument list
buffer_paint		Fill an image buffer with a given colour. buffer *bf Buffer to paint in fpnum *pvec Property vector (i.e. colour)
eigendec		Decompose a symmetric positive semidefinite 2×2 matrix into its eigensystem fpnum *I Input inertia matrix elements stacked row-wise fpnum *D Eigenvalue list fpnum *E Eigenvector matrix elements stacked column-wise
draw_ellipses		Paint a list of blobs as ellipses, sorted with the smallest ellipse on top. buffer *bf_img Background image to paint in buffer *bf_mvec Moment vector list buffer *bf_pvec Property vector list
draw_regions		Paint regions with their average colours. buffer *bf_img Background image to paint in ibuffer *bf_labelim Region label image buffer *bf_pvec Property vector list

Table 12: Methods in visualise.c.

3 Example application

```
/*
**  File: blobdemo_ppm.c
**  Usage: blobdemo_ppm <infile.ppm> <outfile.ppm> [<dmax>]
**  (c) April 2004 Per-Erik Forssen
*/
int main(int argc,char *argv[]) {

fpnum pvec_green[] = {0.0,1.0,0.0}; /* Background colour */
buffer *bf_image,*bf_mvec,*bf_pvec,*bf_blobimage,*bf_rimage;
buffer **bl_lout;
ibuffer *bf_cert,*bf_csc,*bf_labelim;
int regionfl=0;
fpnum testnum;

/* Parameters for the algorithm */
fpnum dmax=0.16;    /* Maximum colour distance */
fpnum cmin=0.5;     /* Area threshold for pyramid generation */
fpnum minc=0.5;     /* Merger threshold */
int roi=0;          /* Side of spatial window (or 0 for 12 pixel roi) */
int miter=5;        /* Number of m-estimation steps */
int lowsc=2;        /* Finest scale to detect blobs in */
int amin=20;        /* Min required area */
int ssfl=0;         /* Subsample image if set */


 if((argc<3)||(argc>5)) {
   fprintf(stderr,"ERROR: At least two filenames should be supplied.\n");
   fprintf(stderr,"Usage: %s <infile.ppm> <outfile.ppm> [<outfile2.ppm>] [<dmax>]\n",ar
   exit(1);
 }

 if(argc==4) {
   testnum=strtod(argv[3],(char **)NULL);
   if(testnum>0) {
     dmax=testnum;  /* Third arg was dmax */
   } else {
     regionfl=1;    /* Third arg was fname */
   }
 }

 if(argc==5) {
   regionfl=1;   /* Third arg was fname */
   dmax=strtod(argv[4],(char **)NULL);
```

```
}

read_pnm_file(argv[1],argv[0],&bf_image,ssfl);

/* Create certainty mask */
bf_cert=ibuffer_new(bf_image->rows,bf_image->cols,bf_image->ndim);
set_to_ones(bf_cert);

/* Allocate array of result pointers */
bl_lout=(buffer **)calloc(4,sizeof(buffer *));

/* Call the blob extraction function */
extract_blobs(bf_image,bf_cert,bl_lout,dmax,cmin,roi,miter,lowsc,minc,amin);

/* Extract results */
bf_mvec    = bl_lout[0];
bf_pvec    = bl_lout[1];
bf_csc     = (ibuffer *)bl_lout[2];
bf_labelim = (ibuffer *)bl_lout[3];

/* Create an empty green image */
bf_blobimage = buffer_new(bf_image->rows,bf_image->cols,bf_image->ndim);
buffer_paint(bf_blobimage,pvec_green);

/* Visualise blobs in the green image */
draw_ellipses(bf_blobimage,bf_mvec,bf_pvec);

/* Store result as a file */
write_pnm_file(argv[2],argv[0],bf_blobimage);

if(regionfl) {
  /* Create an empty green image */
  bf_rimage = buffer_new(bf_image->rows,bf_image->cols,bf_image->ndim);
  buffer_paint(bf_rimage,pvec_green);

  /* Visualise regions in the green image */
  draw_regions(bf_rimage,bf_labelim,bf_pvec);

  /* Store result as a file */
  write_pnm_file(argv[3],argv[0],bf_rimage);
  buffer_free(bf_rimage);
}

/* Free memory */
free(bl_lout);
```

```
 buffer_free(bf_image);
 ibuffer_free(bf_cert);
 buffer_free(bf_blobimage);
 buffer_free(bf_mvec);
 buffer_free(bf_pvec);
 ibuffer_free(bf_csc);
 ibuffer_free(bf_labelim);
 return(0);
}
```

References

[1] Per-Erik Forssén. *Low and Medium Level Vision using Channel Representations*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, March 2004. Dissertation No 858, ISBN 91-7373-876-X.