It's no trick...
it's a vision system

Vision
Components

The Smart Camera People

ColorLib Documentation

Color library for VC cameras Version 2.0

Revision 1.1 01 February 2005
Document name: Color_Lib.pdf
© Vision Components GmbH Ettlingen, Germany

Foreword and Disclaimer

This documentation has been prepared with most possible care. However Vision Components GmbH does not take any liability for possible errors. In the interest of progress, Vision Components GmbH reserves the right to perform technical changes without further notice.

Please notify support@vision-components.com if you become aware of any errors in this manual or if a certain topic requires more detailed documentation.

This manual is intended for information of Vision Component's only. Any publication of this document or parts thereof requires written permission by Vision Components GmbH.

Icons used in this manual

💡 The Light bulb highlights hints and ideas that may be helpful for a development.

⚠ This warning sign alerts of possible pitfalls to avoid. Please pay careful attention to sections marked with this sign.

References:

Description	Titel on Website	Download Area
Installation Manual for VC20XX cameras	InstallVC20XX VC40XX (1.36 MB)	Customer Area ▸Getting Started VC20XX and VC40XX Cameras
Demo programs and sample code used in the manual	Tutorial_Code (14.5 Kbytes)	Customer Area ▸Getting Started VC20XX and VC40XX Cameras
VCRT Operation System Functions Manual	VCRT 5.0 Software Manual (992.31 Kbytes)	Registered User Area ▸Software documentation VC Smart Cameras
VCLIB 2.0 Image Processing Library Manual	VCLIB 2.0 Software Manual (275.42 Kbytes)	Registered User Area ▸Software documentation VC Smart Cameras
VCLIB 3.0 Image Processing Library Manual	VCLIB 3.0 Software Manual (539.67 Kbytes)	Registered User Area ▸Software documentation VC Smart Cameras

© 1996-2005 Michael Engel, Vision Components GmbH Ettlingen, Germany

Table of Contents

© 2005 Vision Components, Ettlingen, Germany

1 Installation

The color library is part of the VCLIB package including `VCLIB.LIB`, `VCLIB.H`, `FLIB.LIB` and `FLIB.H`. If you have already installed VCLIB, no further installation is required. If not, please refer to the installation guide for VCLIB. The color library **does however require a separate software licence**. A valid licence for VCLIB is also required.

It is necessary to call the functions `init_licence()` and `init_vclib()` in this order before programs of the color library can be executed. Both functions require a registration code, which may be part of the delivery or can be obtained from our support team.

It is also possible that you have purchased a licence for a particular camera model. In this case the library will not operate on other camera models. Contact our sales department for registration for other camera models.

If you have the software for evaluation but not a purchased licence, the registration code is only valid for a particular serial number of the camera. Please make sure to include the correct licence codes if you purchase the product later on.

2 Compatibility issues

With the new color library, the image variable structs had to be modified. A `type` field was added, to distinguish between the existing grey value images and the various color image formats. Since color images in the various formats generally have three components instead of only one, two fields (`ccmp1, ccmp2`) were added as start address for the additional color components.

VCLIB 2.0 and 3.0 image variable struct:

```
typedef struct
  {
  long st;              /* start address       */
  int dx;               /* horizontal width    */
  int dy;               /* vertical width      */
  int pitch;            /* memory pitch        */
  } image;
```

VCLIB 3.01 image variable struct:

```
typedef struct
  {
  U8  *st;             /* start address       */
  U32 type;            /* type of image       */
  I32 dx;              /* horizontal width    */
  I32 dy;              /* vertical width      */
  I32 pitch;           /* memory pitch        */
  U8 *ccmp1;           /* color component 1   */
  U8 *ccmp2;           /* color component 2   */
  } image;
```

The choice of the new fields and their location within the new image variable struct are compatible with the previous image variable struct of VCLIB 2.0 and 3.0. Since a field of type `long` requires 64 bits, the 32 leading bits being zero for 32 bit memory addresses, the old convention will always result in the `type` field of the new convention being set to zero.

Compatibility means, that existing projects **do not need to be changed at all**. Projects making use of the new color functions must use the new convention **only for the files where color processing takes place**.

The following shows how to switch between the two conventions:

Typical project file for VCLIB 2.0 and 3.0 convention:

```
#include <register.h>
#include <vcrt.h>
#include <vclib.h>
#include <stdlib.h>
#include <sysvar.h>
#include <macros.h>

. . .
```

Typical project file for VCLIB 3.01 convention:

```
#define  NEW_IMAGE_VAR

#include <register.h>
#include <vcrt.h>
#include <vclib.h>
#include <stdlib.h>
#include <sysvar.h>
#include <macros.h>

. . .
```

`NEW_IMAGE_VAR` is used in the header files `<vclib.h>` and `<macros.h>`, so please make sure to place the definition **before the inclusion of the headers**.

The following table gives an overview of the various possibilities

Existing project, no color functions	No change necessary
Existing project, some color functions need to be added	No change necessary for the files accessing grey images. Files including color processing **must** have `NEW_IMAGE_VAR` defined
Programmer prefers to convert all grey image routines to the new convention	All writes to the struct members `x->st` **must be changed** from `(long)` to `(U8 *)`, `x->type` must be set to 0
New project	We recommend using the new struct for all files (`NEW_IMAGE_VAR` must be defined)
New project with color functions	`NEW_IMAGE_VAR` **must** be defined in all files using color functions. In all other files this is recommended, but not necessary.

3 Display modes

In contrast to “normal” black-and-white cameras, VC color cameras feature various display modes. These display modes are useful to view images stored in the different color image types.

mode	Definition	description
0	DISP_IDLE	no display update, no overhead
1	DISP_GREY	display of grey images, like black-and-white camera
2	DISP_RGB	display of RGB images, image variable type = IMAGE_RGB
3	DISP_BAYER	display of Bayer pattern images, image variable type = IMAGE_BAYER
4	DISP_BAYERGREY	display of Bayer pattern images, image variable type = IMAGE_BAYER, image is shown as grey image
5	DISP_YCBCR	display of YCbCr images, image variable type = IMAGE_CBCR444

The system variable COLOR_MODE is used to change the display modes. With the system variable DISP_PERIOD, the update rate of the display in units of the vertical retrace period can be selected. For color cameras, it is recommended to use somewhat higher values for this variable, resulting in a lower refresh rate, since the refresh of the display must be calculated by the CPU.
Depending on the selected display mode, this may consume a considerable share of CPU time.
It may therefore be desirable to switch the display update off, when maximum CPU time is required for the user program. This may be accomplished by setting either COLOR_MODE or DISP_ACTIVE to zero.

The current camera models store the Bayer pattern from the CCD sensor for image ackquisition.

This is an example of a Bayer pattern. The individual pixels of the CCD
are stored as red (R), green (G) or blue (B) pixel values
Due to the complicated sructure of the array, an interpolation is necessary The current camera models store the Bayer pattern from the CCD sensor for image ackquisition.

R	G	R	G	R
G	B	G	B	G
R	G	R	G	R
G	B	G	B	G
R	G	R	G	R

The storage of a Bayer pattern image requires no more memory than that of a black-and-white image with the same dimensions. It is, however, not very convenient for image processing.
It is therefore recommended, to convert the Bayer pattern image into a different color image type. We suggest using the YCbCr 4:4:4 format, but other formats like RGB or ISH (HIS) will do as well.
YCbCr or RGB can be viewed directly on the screen (COLOR_MODE = DISP_YCBCR or DISP_RGB).
If live display is required, the user may wish to view the original Bayer pattern memory using
COLOR_MODE = DISP_BAYER or DISP_BAYERGREY

4 Important image processing data structures

Gray-scale images / color images

Images and image windows are described by means of so-called image variables, which are described in detail below.
Gray-scale and color images are described using the following image struct:

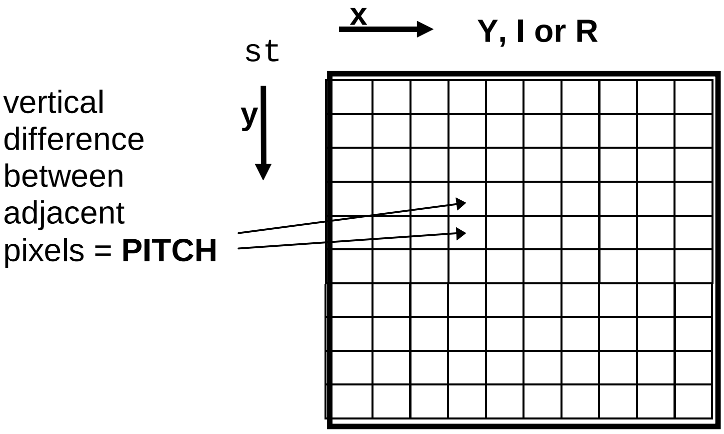
```
typedef struct
  {
  U8  *st;              /* start address       */
  U32 type;             /* type of image       */
  I32 dx;               /* horizontal width    */
  I32 dy;               /* vertical width      */
  I32 pitch;            /* memory pitch        */
  U8 *ccmp1;            /* color component 1   */
  U8 *ccmp2;            /* color component 2   */
  } image;
```

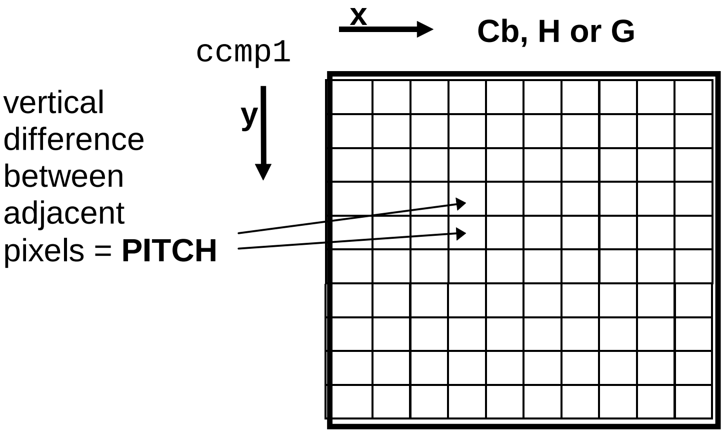
Here, st is the start address of the video memory area, type is one of the following image types, dx and dy are the dimensions of the area of interest in horizontal and vertical direction.

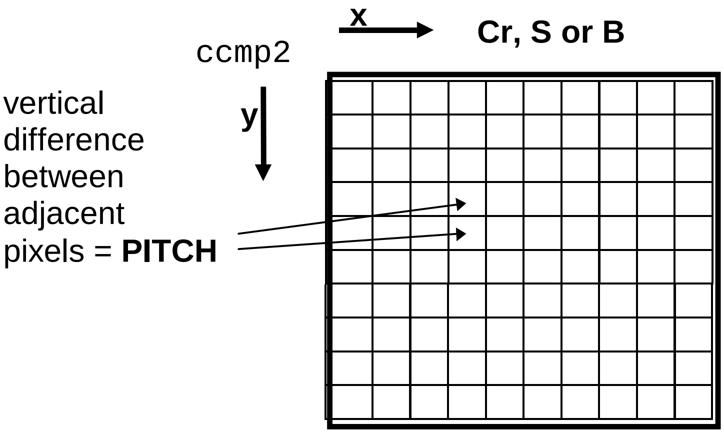
The value pitch is the vertical spacing, i.e. the difference of the address of two vertically adjacent pixels.

ccmp1 and ccmp2 are the start addresses of the additional color components. The following table gives an overview of the various image types and color components.

value	definition	image type	memory requirement	st	ccmp1	ccmp2
0	IMAGE_GREY	grey-scale image U8	dy * pitch	grey	---	---
1	IMAGE_BAYER	Bayer pattern	dy * pitch	bayer	---	---
2	IMAGE_RGB	color image RGB	3 * dy * pitch	red	green	blue
3	IMAGE_CBCR444	color image YCbCr 4:4:4	3 * dy * pitch	y	u	v
4	IMAGE_CBCR411	color image YCbCr 4:1:1	3/2* dy * pitch	y	u	v
5	IMAGE_YUVNORM	normalized YcbCr 4:4:4	3 * dy * pitch	y	u*	v*
6	IMAGE_IHS	color image IHS (HSI)	3 * dy * pitch	i	h	s







For a grey scale image, the upper left pixel is stored at address `st`. Going 1 to the right (x-direction), 1 must be added to this address. Going down (y-direction), `pitch` must be added. For a color image, two additional memory blocks are needed to store the color information. It makes sense to store these 2 blocks tightly behind the first block and behind each other, but this is not guaranteed. For instance, consider the case when the image variable describes a subframe of another frame. In this case, additional pointers are necessary (`ccmp1` and `ccmp2`)

5 Additional Shell Commands for Color Cameras

The shell contains the following additional internal commands:

wb	white balance	`wb`
disp	change display mode	`disp [<option>]`

wb **white balance**

synopsis `wb`

description The command `wb` performs a white balance for color cameras. It is not available for black-and-white cameras and all cameras with the serial number of a black-and-white camera but have a color head as a special option.

Procedure:

1. The user enters `wb`
2. The shell responds with:

```
Please place white object inside yellow frame
and select a brightness between 100 and 180
Press any key for start and end
```

3. The camera enters the interactive mode and displays the average grey value of the region inside the yellow overlay frame.
4. Place a white or grey (colorless) object (e.g. a piece of paper) under the camera covering the complete area inside the yellow overlay frame
5. Adjust brightness (iris of the lens, illumination) so that the average brightness displayed is between the limits (100 and 180). If the values are higher, the values for RGB might be saturated. If the values are lower, the white balance might be inaccurate.
6. If step 5 is not possible, hit a key to exit the interactive mode. Change the shutter setting with the `sh` – command and repeat steps 1-5.
7. Press any key to exit the interactive mode. The white balance values are calculated, output on the console, stored as system variables (`RED, GREEN, BLUE`) and the input color lookup table is programmed.
8. If you type `vd` after the shell's $-promt to get a live image, you will notice that the tint of the image has changed.

disp **change display mode**

synopsis `disp [<option> <number>]`

description The command `disp` changes the display mode. There are several options, some of which are not available for black-and-white cameras:

-c	change color mode	(color only)
-g	change gamma correction	
-p	change display period	

option –c:

This option changes the color mode for the display. Images can be displayed in a variety of color formats including grey value output (black-and-white) and YUV format (YCbCr)

`0`	`IDLE`
`1`	`GREY`
`2`	`RGB`
`3`	`BAYER`
`4`	`BAYERGREY`
`5`	`YCBCR`

example `disp -c 5` change to YCbCr display

option –g:

This option allows to set the gamma correction for the display. Display monitors normally have a non-linear, mostly logarithmic transfer function. You can enter 100 times gamma with this command.

example `disp -g 100` change gamma to 1 (default is 0.6)

option –p:

This option changes the refresh rate (`DISP_PERIOD`) of the display. Display refresh adds a certain overhead, which slows down the processing power of the CPU. For black-and-white cameras, this overhead is mostly negligible, since only memory transfers are involved, the CPU running at full speed. For color cameras, however, the CPU must calculate the color conversion, which is quite time consuming. A color conversion may take up to 60 milliseconds depending on color mode and DSP type and speed grade. The refresh rate is defined in units of the vertical retrace time which is typically 14 milliseconds for an SVGA display. This command also changes the system variable `DISP_PERIOD`.

example `disp -p 10` change refresh rate to 140 milliseconds

6 Macros

The file `macros.h` contains macros that are useful for working with the library. It is not necessary to use these macros, but it may turn out to be convenient.
The following types of macros are available:

- definition of bits, bytes, words, pages
- aliases for video modi
- conversion macros
- image variable macros
- screen macros
- overlay macros
- utility macros

Some macros (screen macros) use conventions for physical and logical addresses. There is, again, no obligation to use these conventions and the according macros.

When using the macros for color image variables, `NEW_IMAGE_VAR` must be defined **before** macros.h is included.

assignment of a whole image variable in just one statement

```
#define ImageAssign(a,newst,newdx,newdy,newpitch)
{
(a)->st=(U8 *)(newst);
(a)->type=0;
(a)->dx=(I32)(newdx);
(a)->dy=(I32)(newdy);
(a)->pitch=(I32)(newpitch);
(a)->ccmp1=(U8 *)0;
(a)->ccmp2=(U8 *)0;
}
```

In comparison to the previous definition, the 3 new members of the new image variable are assigned a value of 0. This corresponds to an image variable definition of a grey (black-and-white) image. This means, invovation of this macro gives the same results as the previous version, but it was written using the new convention.

```
#define ImageAssignC(a,newst,newtype,newdx,newdy,newpitch,
                                               newccmp1,newccmp2)
{
(a)->st=(U8 *)(newst);
(a)->type=(newtype);
(a)->dx=(I32)(newdx);
(a)->dy=(I32)(newdy);
(a)->pitch=(I32)(newpitch);
(a)->ccmp1=(U8 *)(newccmp1);
(a)->ccmp2=(U8 *)(newccmp2);
}
```

This is a new macro for assigning color image variables. The new members `type`, `ccmp1` and `ccmp2` can be assigned with this macro.

7 Sample image variables

1. The pattern of a part is to be stored in a gray image with the size 256(h) x 128(v).

```
#define  NEW_IMAGE_VAR
#include <vclib.h>
#include <macros.h>

main()
{
image a =   {(U8 *)0,              /* start address           */
                   0,              /* type = 0: IMAGE_GREY    */
                 256,              /* dx                      */
                 128,              /* dy                      */
                 256,              /* pitch                   */
             (U8 *)0,              /* ccmp1                   */
             (U8 *)0,              /* ccmp2                   */

ImageAssign(&a, getvar(CAPT_START), 256, 128, 256)
a.st = (long)(getvar(CAPT_START)); /* assign start of image   */
                                   /* to address of capture   */
                                   /* memory buffer           */
...
```

Selecting 256 for pitch produces a *tight* version of the image in memory, without gaps. This is not always the case. When pictures are taken, the resulting image sometimes contains gaps, meaning that `pitch` is greater than dx. However, `pitch` may **never be smaller than dx**.

2. A full frame (a) is assumed to have a size of 640(h) x 480(v) with a pitch of 640. Two partial images (b, c) with a size of 128(h) x 128(v) are to be defined in this full frame. The partial images will later be used to evaluate the image.

```
#define  NEW_IMAGE_VAR
#include <vclib.h>
#include <macros.h>

main()
{
image a, b, c;

ImageAssign(&a, getvar(CAPT_START), 640, 480, getvar(VPITCH));
ImageAssign(&b, a.st+100*getvar(VPITCH)+200, 128, 128, getvar(VPITCH));
ImageAssign(&c, a.st+200*getvar(VPITCH)+300, 128, 128, getvar(VPITCH));

. . .
```

The upper left corner of the image window b is located at position (200,100) of the full frame a. The upper left corner of the image window c is located at position (300,200).

If it is desired, for example to set the contents of the image variable c to the constant value 255 (white), this can be done with the following function call:

```
set(&c,255);
```

3. Same as 2. but for a color image variable of type `IMAGE_CBCR444`. The addresses for the 2 additional components (chrominance components) need to be calculated in this example. In this case, this is done by calling the function `ImageAllocate()` which allocates memory for this particular type of image variable and assigns the proper values to the image variable. This is a difference to the previous example, since example 2 doesn't allocate memory, it just uses the memory already allocated for capture purposes.

```
#define  NEW_IMAGE_VAR
#include <vclib.h>
#include <macros.h>

main()
{
image a, b, c;

ImageAllocate(&a, IMAGE_CBCR444, 640, 480);
a.pitch = getvar(VPITCH));

ImageAssignC (&b, IMAGE_CBCR444,
      a.st+100*getvar(VPITCH)+200, 128, 128, getvar(VPITCH),
      a.ccmp1+100*getvar(VPITCH)+200,
      a.ccmp2+100*getvar(VPITCH)+200);

ImageAssignC(&c, IMAGE_CBCR444,
      a.st+200*getvar(VPITCH)+300, 128, 128, getvar(VPITCH),
      a.ccmp1+200*getvar(VPITCH)+300,
      a.ccmp2+200*getvar(VPITCH)+300);


. . .
```

As in the previous example, the upper left corner of the image window b is located at position (200,100) of the full frame a. The upper left corner of the image window c is located at position (300,200).

If it is desired, for example to set the contents of the color image variable c to the constant value 128 (medium grey, no color), this can be done with the following function call:

```
cset(&c,128, 0, 0);
```

8 Programs for processing color images

init_licence	initialize licence code
init_vclib	initialize color library
ImageAllocate	memory allocation for an image variable
ImageFree	release memory for an image variable
cset	set color image variable to a constant value
copy	copy an image variable
fwrite_image	write bitmap image to file
fread_image	read bitmap image from file
ColorBar	color bar test chart
ColorGraph	color graph test chart
WhiteBalanceValues	calculate white balance values
init_color_lut	initialize color input LUT
init_color_table	initialize color software lookup-table
clut_bayer	bayer color lookuptable operation
init_LUT_gamma	init image output LUT using gamma correction
BayerToGrey	Bayer Pattern to Grey conversion
BayerToRGB	Bayer Pattern to RGB conversion
BayerToYCbCr	Bayer Pattern to YCbCr conversion
RGB_YCbCr	RGB to YCbCr color conversion
YCbCr_RGB	YCbCr to RGB color conversion
YCbCr_NORM	YCbCr to normalized YCbCr conversion
NORM_YCbCr	normalized YCbCr to YCbCr conversion
RGB_IHS	RGB to IHS (HSI) color conversion
color_histo	color histogram of a color image variable
display_chisto	display color histogram
color_classify	color classification

standard error returns

Most of the functions return a standard error code:

```
#define ERR_NONE         0   /* no error                */
#define ERR_FORMAT      -1   /* image format error      */
#define ERR_TYPE        -2   /* image type error        */
#define ERR_MEMORY      -3   /* out of memory           */
#define ERR_LICENCE     -5   /* licence required        */
#define ERR_OPEN       -19   /* open error              */
#define ERR_MODEL       -51  /* model does not fit licence */
```

init_licence — initialize licence code

synopsis

```
I32 init_licence (char *code)
```

description ⚠

This function initializes the VCLIB and other special VC Libraries.
This function must be called prior to using any VCLIB functions or other special library functions.

The function returns 0 on proper initialization, negative numbers on error.

possible error codes:

```
ERR_LICENCE      /* licence required          */
ERR_OPEN         /* open error                */
ERR_MODEL        /* model does not fit licence */
```

example

```
init_licence(”T1122334455”)  /*initializing a full VCRT/
VCLIB Licence for VC cameras with Texas Instrument DSP*/

init_licence(”C1122334455”)  /*initializing a full Color
LIB licence for VC cameras with Texas Instrument DSP*/
```

memory

none

explanation

Vision Components continues to offer special libraries to their customers. For simpler handling and ensured compatibility all libraries are now included in one setup package – for instance: TI-VCRT523_VCLIB300_Setup.exe

In order to use any VCLIB, ColorLIB or other special VC Library functions, each library requires initialization prior to using its functions.

The VCLIB Licence code is displayed on the delivery docket and user CD shipped with the delivery of the VC SDK-T[1].
Licence codes for other special libraries as the ColorLIB are also issued on delivery notes or emailed. Please contact sales@vision-comp.com for a quote on development software and special libraries from Vision Components.

licence types

The following licence types are currently available / under preparation:

T	Full Licence for programming all VC cameras with TI DSP
M	Full Licence of the M200 Data Matrix Code Reader Library
C	Full Licence of the ColorLib
E	Full Licence of the Extension Lib
L	Loan Licence valid for 3 month only (in combination with T,L,C or E)
P	Licence restricted to VC4018 smart cameras (in c. .with T,L,C or E)
Q	Licence restricted to VC4038 smart cameras (in c. .with T,L,C or E)

[1] Vision Components Software Devellopment Kit for Smart Cameras with Texas Instrument DSP, containing the TI C-cross compiler and VCRT and VCIB Libraries from Vision Components.

ImageAllocate **memory allocation for an image variable**

synopsis `U8 *ImageAllocate(image *img, U32 type, U32 dx, U32 dy)`

description This function allocates memory for an image and sets the image variable struct components to the appropriate values.

`type` is the image type, `dx` and `dy` are the horizontal and vertical dimensions of the image. The function allocates memory for a tight storage, i.e. `img->pitch` is set to `dx`.

value	**type**	**image type**	**memory requirement**	**st**	**ccmp1**	**ccmp2**
0	IMAGE_GREY	grey-scale image U8	dx * dy	grey	---	---
1	IMAGE_BAYER	Bayer pattern	dx * dy	bayer	---	---
2	IMAGE_RGB	color image RGB	3 * dx * dy	red	green	blue
3	IMAGE_CBCR444	color image YCbCr 4:4:4	3 * dx * dy	y	u	v
4	IMAGE_CBCR411	color image YCbCr 4:1:1	3/2 * dx * dy	y	u	v
5	IMAGE_YUVNORM	normalized YCbCr 4:4:4	3 * dx * dy	y	u*	v*
6	IMAGE_IHS	color image IHS (HSI)	3 * dx * dy	i	h	s

The function returns the start address of the memory block allocated. If out of memory, the NULL pointer is returned.

memory see table

see also `ImageFree()`

ImageFree **release memory for an image variable**

synopsis `void ImageFree(image *img)`

description This function frees the memory for an image previously allocated with `ImageAllocate()`.

memory none

see also `ImageAllocate()`

cset — set color image variable to a constant value

synopsis `I32 cset(image *rgb, I32 x, I32 y, I32 z)`

description The function `cset()` sets all pixels of a color image variable to the constant values x (first color component), y (second color component) and z (third color component).

The function may be used for the following color image types:

`IMAGE_RGB`	`x: red,`	`y: green,`	`z: blue`
`IMAGE_CBCR444`	`x: Y,`	`y: cb,`	`z: cr`
`IMAGE_CBCR411`	`x: Y,`	`y: cb,`	`z: cr`
`IMAGE_IHS`	`x: i,`	`y: h,`	`z: s`

The function returns the standard error code.

memory none

see also `set()`

copy — copy an image variable

synopsis `I32 copy(image *src, image *dst)`

description The function `copy` copies the contents of the image variable src to dst.

If the format of the image variable (dx, dy) is not identical, the format of the result variable `dst` is used. In particular, this means that the result of the operation is not defined if the image format of `src` is smaller than that of `dst`. (`src->dx < dst->dx or src->dy < dst->dy`)

You are recommended to work with identical image formats, i.e. `src->dx = dst->dx` and `src->dy=dst->dy`

It is possible to copy the contents of color images with this function. In this case, the copy is performed only, if `src` and `dst` belong to the same storage classes, i.e. require the same amount of memory. I.e. a copy from `type = IMAGE_RGB` to `IMAGE_IHS` is allowed, whereas a copy from `IMAGE_GREY` to `IMAGE_RGB` is not allowed. It is recommended to use images of the same type for `src` and `dst`.

The function returns the standard error code.

memory none

fwrite_image — write image variable as bit map file (BMP)

synopsis `I32 fwrite_image (char *path, image *img)`

description This function writes the image defined by **image variable** `img` as a bit map file (BMP) to the file specified by `path`.

Currently, the only image type supported is `IMAGE_RGB`. Images are stored in 24 bit true-color mode.

The function returns the standard error code.

memory none

see also `fread_image()`

fread_image — read a bit map file (BMP) and write to image variable

synopsis `I32 fread_image (char *path, image *img)`

description This function reads the bit map image (BMP) from the file specified by path and stores it in **image variable** `img`.

Currently, the only image type supported is `IMAGE_RGB`. BMP images must be stored in 24 bit true-color mode.

If the BMP image is larger than the image variable, the image size is truncated to the size of the image variable.

memory none

see also `fwrite_image()`

ColorBar — color bar test chart

synopsis

```
I32 ColorBar (image *rgb, U32 amplitude, U32 saturation)
```

description

This function creates a color bar test chart with vertical bars of white, yellow, cyan, green, magenta, red, blue, black colors(from left to right). `amplitude` and `saturation` specify the values for the color bar amplitude and saturation. Allowed values range from 0 (zero amplitude, saturation) to 255 (maximum amplitude, saturation). The following tables may be helpful:

100% amplitude, 100% saturation
`amplitude = 255, saturation = 255`

	white	yellow	cyan	green	magenta	red	blue	black
R	255	255	0	0	255	255	0	0
G	255	255	255	255	0	0	0	0
B	255	0	255	0	255	0	255	0
Y	**255**	**226**	**179**	**150**	**105**	**76**	**29**	**0**
Cb	**128**	**0**	**171**	**44**	**212**	**85**	**255**	**128**
Cr	**128**	**149**	**0**	**21**	**235**	**255**	**107**	**128**

75% amplitude, 75% saturation
`amplitude = 191, saturation = 191`

	white	yellow	cyan	green	magenta	red	blue	black
R	191	191	47	47	191	191	47	47
G	191	191	191	191	47	47	47	47
B	191	47	191	47	191	47	191	47
Y	**191**	**175**	**148**	**132**	**107**	**90**	**63**	**47**
Cb	**128**	**56**	**152**	**80**	**176**	**104**	**200**	**128**
Cr	**128**	**140**	**56**	**68**	**188**	**200**	**116**	**128**

The image variable `rgb` may currently be of type `IMAGE_RGB` or `IMAGE_CBCR444.`

The function returns the standard error code.

see also `ColorGraph()`

© 2005 Vision Components, Ettlingen, Germany

ColorGraph

color graph test chart

synopsis

```
I32 ColorGraph (image *dst, I32 y, I32 sat,
                      I32 mode, float start_color)
```

description

This function creates a color graph test chart showing all colors with constant luminance y and constant saturation sat.

The colors are stored in the image described by **image variable** dst.
They can be stored horizontally (i.e. different colors moving from left to right, but constant colors moving vertically) or vertically depending on the value of the parameter mode. A start color can be defined by start_color ranging from 0 to 2π.
Allowed values for y , sat range from 0 (zero amplitude, saturation) to 255 (maximum amplitude, saturation).

The image variable rgb may currently be of type IMAGE_RGB, IMAGE_CBCR444 or IMAGE_YUVNORM

The function returns the standard error code.

see also

ColorBar()

© 2005 Vision Components, Ettlingen, Germany

WhiteBalanceValues calculate white balance values

synopsis

```
I32 WhiteBalanceValues (image *bayer,
            I32 *red, I32 *green, I32 *blue)
```

description

This function performs the calculation of the white balance values for red, green and blue.

`bayer` should be an image variable of type `IMAGE_RGB.` `Red`, `green` and `blue` are pointers for the storage of the white balance correction values calculated by the function.

`bayer.st` must point to a red pixel, i.e. there must be an even number of pixels in horizontal and vertical direction between the start of the captured image and the `bayer.st.`

The return value of the function is the maximum of the average intensities for the red green and blue pixels.

Before the function is executed, the input lookup-table must be set to equal amplification for the 3 channels, for example with the statement:

```
init_color_lut(1024, 1024, 1024);
```

A white or grey reference image must be presented to the camera for the pixels specified by the image variable `bayer` .

The result of this function is only reliable, if the pixel intensity is within a specific range. For this reason, the function outputs the maximum average red/green/blue intensities. This value should be in the range of [100..180].
If the value is less than 100, the image might be too noisy and the resolution will suffer. If the value is higher than 180, some of the pixel might be saturated, which will result in an incorrect white balance.

memory

none

see also

`init_color_lut()`

init_color_lut **initialize color input LUT**

synopsis

```
void init_color_lut (I32 red, I32 green, I32 blue)
```

description

This function programs the **hardware** input color lookup-table to a linear mapping between input and output.
The mappings for the red, green and blue channels can be programmed to a different slope, which is a useful feature for adjusting the whitebalance of the camera.

Slope values for `red`, `green` and `blue` can be used to amplify each channel (value > 1024) or attenuate the channel (value < 1024). A value of 1024 will result in an identity transform.

9 bits are used for the input of the LUT, 8 bits for the output, so there is enough head-room for some amplification.

For the whitebalance adjustment, we recommend to leave the channel with the maximum intensity at the identity transform, the other two channels should be amplified by appropriate factors.

The possible range for `red`, `greeen` and `blue` is [0.. 32768] equivalent to amplification factors between 0 and 32.

side effects

The function changes the values of the system variables `RED`, `GREEN` and `BLUE`.

memory

`none`

see also

`WhiteBalanceValues()`, `init_color_table()`

init_color_table initialize color software lookup-table

synopsis

```
I32 init_color_table (U32 red,
           U32 green, U32 blue, U8 table[])
```

description

This function programs a **software** color lookup-table to a linear mapping between input and output.
The mappings for the red, green and blue channels can be programmed to a different slope, which is a useful feature for adjusting the whitebalance of the camera.

Slope values for `red`, `green` and `blue` can be used to amplify each channel (value > 1024) or attenuate the channel (value < 1024). A value of 1024 will result in an identity transform.

8 bits are used for the input of the LUT, 8 bits for the output.

This function may be used for whitebalance adjustment for all cameras without hardware input LUT.

For the whitebalance adjustment, we recommend to leave the channel with the maximum intensity at the identity transform, the other two channels should be amplified by appropriate factors.

The possible range for `red`, `greeen` and `blue` is [0.. 32768] equivalent to amplification factors between 0 and 32.

side effects

The function changes the values of the system variables `RED`, `GREEN` and `BLUE`.

memory

`none`

see also

`WhiteBalanceValues(), init_color_lut(), clut_bayer()`

clut_bayer **bayer color lookuptable operation**

synopsis `I32 clut_bayer (image *bayer1, image *bayer2, U8 table[])`

description This function performs a bayer color lookuptable operation for all cameras without hardware input LUT.

`bayer1` is the image variable for the source image, `bayer2` for the destination image. Both images must be of `type = IMAGE_BAYER`. The image variable `bayer1` should start with a red pixel.
`table[1024]` is the lookup-table which contains the mapping of the input pixels to the output pixels. `table[]` consists of 4 independent LUTs with 256 values each for R1, G1, G2 and B2 in this sequence.

R1	G1	R1	G1	R1
G2	B2	G2	B2	G2
R1	G1	R1	G1	R1
G2	B2	G2	B2	G2
R1	G1	R1	G1	R1

The output of this function is a bayer pattern with the same organization as the input image with an individual mapping of red, green and blue pixels.

`table` may be the a linear mapping table calculated with `init_color_table()`, e.g. for a whitebalance operation

The function returns the standard error code.

see also `init_color_lut()`

init_LUT_gamma **init image output LUT using gamma correction**

synopsis `void init_LUT_gamma(float gamma)`

description This function programs the image output lookuptable (output LUT) for black-and-white / color display using gamma correction.

Gamma correction is a non-linear function used in order to compensate for display monitor non-linearities.

The following formula is applied:

X’ = X ^ gamma , where X may be any of R,G,B

Higher values for gamma tend to increase contrast while at the
same time low grey values (dark areas) may not be distinguishable.
Lower values decrease contrast and dark areas may be better differentiated.

© 2005 Vision Components, Ettlingen, Germany

The standard value for gamma is 0.45 (according to various video standards).
We recommend a value of 0.6 .
Of course, the best value depends on the chosen monitor and its settings (like brightness and contrast) and may be found using some experimentation.

see also `init_LUT()`

© 2005 Vision Components, Ettlingen, Germany

BayerToGrey

Bayer Pattern to Grey conversion

synopsis

```
I32 BayerToGrey (image *bayer, image *grey)
```

description

This function converts Bayer pattern images (`type = IMAGE_BAYER`) to grey images (`type = IMAGE_GREY`).

`bayer` specifies the **image variable** for the input Bayer pattern image, `grey` for the resulting output image.

The routine uses a 5x5 filter mask for maximum resolution for the resulting grey value image.

The following simple formula is applied to convert rgb-values to grey:

```
y = r + 2*g + b
```

This means that the resulting grey value is not absolutely physiologically correct. For Machine Vision, however, it is a good choice.

The function returns the standard error code.

see also `BayerToRGB(), BayerToYCbCr()`

BayerToRGB

Bayer Pattern to RGB conversion

synopsis

```
I32 BayerToRGB (image *bayer, image *rgb)
```

description

This function converts Bayer pattern images (`type = IMAGE_BAYER`) to RGB type images (`type = IMAGE_RGB`).

`bayer` specifies the **image variable** for the input Bayer pattern image, `rgb` for the resulting output image.

The routine uses a 5x5 filter mask for maximum resolution for the resulting grey value image.

The function returns the standard error code.

see also `BayerToGrey(), BayerToYCbCr()`

BayerToYCbCr — Bayer Pattern to YCbCr conversion

synopsis

```
I32 BayerToYCbCr (image *bayer, image *ycbcr)
```

description

This function converts Bayer pattern images (type = IMAGE_BAYER) to YCbCr type images (type = IMAGE_CBCR444).

bayer specifies the **image variable** for the input Bayer pattern image, ycbcr for the resulting output image.

The routine uses a 5x5 filter mask for maximum resolution for the resulting grey value image.
We strongly recommend using this particular function instead of using BayerToRGB() since the YcbCr format is compatible with most of the VCLIB functions for grey value images.

The function returns the standard error code.

see also BayerToGrey(),BayerToRGB()

RGB_YCbCr — RGB to YCbCr color conversion

synopsis

```
I32 RGB_YCbCr (image *rgb, image *ycbcr)
```

description

This function converts RGB images to YCbCr images using the following formula:

```
Y  =  0.29900 * R + 0.58700 * G + 0.11400 * B
Cb = -0.16874 * R - 0.33126 * G + 0.50000 * B  + 128
Cr =  0.50000 * R - 0.41869 * G - 0.08131 * B  + 128
```

rgb specifies the **image variable** for the input RGB image, ycbcr for the resulting YCbCr output image. All results are rounded properly using 4/5 rounding.

The function returns the standard error code.

The image variable rgb must be of type IMAGE_RGB, ycbcr must be of type IMAGE_CBCR444. The function may be called in-place, i.e. with the same image variable pointer for rgb and ycbcr. In this case, the image variable must be of type IMAGE_CBCR444.

see also YCbCr_RGB()

YCbCr_RGB — YCbCr to RGB color conversion

synopsis

```
I32 YCbCr_RGB (image *ycbcr, image *rgb)
```

description

This function converts YCbCr images to RGB images using the following formula:

```
R = Y                          + 1.40200 * (Cr – 128)
G = Y - 0.34414 * (Cb – 128) - 0.71414 * (Cr – 128)
B = Y + 1.77200 * (Cb – 128)
```

`ycbcr` specifies the **image variable** for the input YCbCr image, `rgb` for the resulting RGB output image. All results are rounded properly using 4/5 rounding.

The function returns the standard error code.

The image variable `ycbcr` must be of type `IMAGE_CBCR444`, `rgb` must be of type `IMAGE_RGB`. The function may be called in-place, i.e. with the same image variable pointer for `ycbcr` and `rgb`. In this case, the image variable must be of type `IMAGE_CBCR444`.

see also `RGB_YCbCr()`

YCbCr_NORM — YCbCr to normalized YCbCr color conversion

synopsis

```
I32 YCbCr_NORM (image *ycbcr, image *ynbnr)
```

description

This function converts YCbCr images to normalized YCbCr images using the following formula:

```
Nb = CB*(Cb – 128) / Y + 29
Nr = CR*(Cr – 128) / Y + 76
```

with:

```
CB =  51.56620
CR = 106.86900
```

The function operates on the color components only, i.e. it leaves the Y component unchanged.

`ycbcr` specifies the **image variable** for the input YCbCr image, `ynbnr` for the resulting normalized YCbCr output image. All results are rounded properly using 4/5 rounding.

The function returns the standard error code.

The image variable ycbcr must be of type IMAGE_CBCR444, ynbnr must be of type IMAGE_YUVNORM. The function may be called in-place, i.e. with the same image variable pointer for ycbcr and ynbnr. In this case, the image variable must be of type IMAGE_YUVNORM.

see also NORM_YCbCr()

NORM_YCbCr normalized YCbCr to YCbCr color conversion

synopsis

```
I32 NORM_YCbCr (image * ynbnr, image *ycbcr)
```

description

This function converts normalized YCbCr images to YCbCr images using the following formula:

```
Nb = XB*(Nb – 29) * Y + 128
Nr = XR*(Nr – 76) * Y + 128
```

with:

```
XB = 0.019317
XR = 0.009357
```

The function operates on the color components only, i.e. it leaves the Y component unchanged.

`ynbnr` specifies the **image variable** for the input normalized YCbCr image, `ycbcr` for the resulting YCbCr output image. All results are rounded properly using 4/5 rounding.

The function returns the standard error code.

The image variable `ycbcr` must be of type `IMAGE_CBCR444`, `ynbnr` must be of type `IMAGE_YUVNORM`. The function may be called in-place, i.e. with the same image variable pointer for `ycbcr` and `ynbnr`. In this case, the image variable must be of type `IMAGE_CBCR444`.

see also `YCbCr_NORM()`

RGB_IHS RGB to IHS (HSI) color conversion

synopsis `I32 RGB_IHS (image *rgb, image *ihs)`

description This function converts RGB images to IHS using the following formula:

define:

```
MIN = min(R, G, B)
MAX = max(R, G, B)
DELTA = MAX - MIN
```

then:

```
I = (R + 2*G + B)/4
S = (I - MIN)/I

R == max:   H =       (85*(G-B)) / (2*DELTA);        (1)
G == max:   H =  85 + (85*(B-R)) / (2*DELTA);        (2)
B == max:   H = 170 + (85*(R-G)) / (2*DELTA);        (3)

(1)   /* -42.5 to  42.5 between yellow  & magenta */
(2)   /*  42.5 to 127.5 between cyan    & yellow  */
(3)   /* 127.5 to 212.5 between magenta & cyan    */
```

`rgb` specifies the image variable for the input RGB image, `ihs` for the resulting IHS output image.
The function returns the standard error code.

The image variable `ycbcr` must be of type `IMAGE_CBCR444`, `rgb` must be of type `IMAGE_RGB`. The function may be called in-place, i.e. with the same image variable pointer for `ycbcr` and `rgb`. In this case, the image variable must be of type `IMAGE_CBCR444`.

see also `RGB_YCbCr()`

color_histo color histogram of a color image variable

synopsis `I32 color_histo (image *img, U32 hist[65536])`

description The function `histo` calculates the color histogram of the color mage variable img. Supported image types are:

```
IMAGE_CBCR444
IMAGE_YUVNORM
IMAGE_IHS
```

The color histogram is calculated for the color components of the image only. I.e. the luminance signal for YcbCr and YUVnorm and the Intensity for the HIS color model **are not considered for the calculation.**

The histogram is the absolute frequency of the 65536 different colors (different hue, different saturation) in an image/image window.
In addition to the color image variable img, a pointer to the histogram array with 65536 values is passed to the function. After calling the function, the result can be taken from this array.

Since `hist[]` is a two-dimensional array of 256x256 values, it is important to know, how to address it properly:

```
Histvalue = hist[v*256 + u])
```

So, the u-component (Cb, H) is stored as first index, the v-component (Cr, S) as second index.

The function returns the standard error code.

memory none

see also `display_chisto()`

display_chisto display color histogram

synopsis `I32 display_chisto (image *map, U32 hist[65536])`

description The function `display_chisto` generates a two-dimensional plot of the `hist` array.
The image described by the image variable `map` must have at least 256x256 pixels since the plot will produce exactly 256x256 pixels.
The type of the image variable `map` may be

```
IMAGE_GREY
IMAGE_RGB
IMAGE_CBCR444
```

© 2005 Vision Components, Ettlingen, Germany

The function displays the the first component of the histogram (Cb, u, H) in the horizontal direction, the second components (Cr, v, S) is displayed vertically.
If the result image `map` is a color image type, the display is done with the color of the corresponding original image color, but with a constant saturation.

The function returns the standard error code.

memory `none`

see also `color_histo()`

© 2005 Vision Components, Ettlingen, Germany

color_classify — color classification

synopsis

```
I32 color_classify (image *src, image *dst, U8 table[])
```

description

`color_classify` performs a color classification of a **color image variable**. The following types are allowed for the source **image variable** `src`:

```
IMAGE_CBCR444
IMAGE_YUVNORM
IMAGE_IHS
```

The function works on the color components (Cb, Cr / HS) of the **image variable** only. The luminance (Y / I) is **not** considered for this operation

For each individual pixel, the color components act as a two-dimensional index to the classification array `table[]` and the table value is output for the destination image `dst`.

`dst` must be an image of type `IMAGE_GREY`

`table[]` is an array with 256x256=65536 values.

The function returns the standard error code.

example

Assume that we have a color image of type `IMAGE_CBCR444`. Assume further that we want to select all exactly colorless pixels, i.e. `Cb=Cr=0`.
Since `Cr` and `Cb` are stored with offset 128, set

```
table[128][128] = 1
table[u][v]     = 0 for u,v != 128
```

The destination image will have value 1 for all absolutely colorless pixels with Cb=128, Cr=128. All other pixels will have value 0.

see also

`color_histo()`

Appendix A: List of library functions

Color library functions

Name	Type	Description
`I32 init_licence(char *code)`	C	initialize licence code
`U8 *ImageAllocate (image *img, U32 type, U32 dx, U32 dy)`	C	memory allocation for an image variable
`void ImageFree (image *img)`	C	release memory for an image variable
`I32 cset (image *rgb, I32 x, I32 y, I32 z)`	C	set color image variable to a constant value
`I32 copy (image *src, image *dst)`	C	copy an image variable
`I32 fwrite_image(char *path, image *img)`	C	write image variable as a bit map file (BMP)
`I32 fread_image(char *path, image *img)`	C	read a bit map file (BMP) and write to image variable
`I32 ColorBar(image *rgb, U32 y, U32 sat)`	C	color bar test chart
`I32 ColorGraph(image *dst, I32 y, I32 sat, I32 mode, float start_color)`	C	color graph test chart
`I32 WhiteBalanceValues(image *bayer, I32 *red, I32 *green, I32 *blue)`	C	calculate white balance values
`void init_color_lut(I32 red, I32 green, I32 blue)`	C	initialize color input LUT
`I32 init_color_table(U32 red, U32 green, U32 blue, U8 table[])`	C	initialize color software lookup-table
`I32 clut_bayer(image *bayer1, image *bayer2, U8 table[])`	C	bayer color lookuptable operation
`void init_LUT_gamma (float gamma)`	C	output LUT gamma correction
`I32 BayerToGrey(image *bayer, image *grey)`	C	Bayer Pattern to Grey conversion

Name	**Type**	**Description**
`I32 BayerToRGB(image *bayer, image *rgb)`	C	Bayer Pattern to RGB conversion
`I32 BayerToYCbCr(image *bayer, image *ycbcr)`	C	Bayer Pattern to YCbCr conversion
`I32 RGB_YCbCr(image *rgb, image *ycbcr)`	C	RGB to YCbCr color conversion
`I32 YCbCr_RGB(image *ycbcr, image *rgb)`	C	YCbCr to RGB color conversion
`I32 YCbCr_NORM(image *ycbcr, image *ynbnr)`	C	YCbCr to normalized YCbCr color conversion
`I32 NORM_YCbCr(image * ynbnr, image *ycbcr)`	C	normalized YCbCr to YCbCr color conversion
`I32 RGB_IHS(image *rgb, image *ihs)`	C	RGB to IHS (HSI) color conversion
`I32 color_histo(image *img, U32 hist[])`	C	color histogram of a color image variable
`I32 display_chisto(image *map, U32 hist[])`	C	display color histogram
`I32 color_classify(image *src, image *dst, U8 table[])`	C	color classification

Legend: A: Assembly function C: C function M: Macro

INDEX

© 2005 Vision Components, Ettlingen, Germany