VisionCLIPS

Users guide

Guido Appenzeller
Marianne Hardt
Cordelia Schmid

December 8, 1994
Contents

1 Introduction ............................................. 4
   1.1 Basic concepts .................................... 4
   1.2 CLIPS ............................................. 4
   1.3 Overview of existing functions .................... 4
   1.4 Static and dynamic linking ......................... 4

2 Modules for VisionCLIPS .......................... 5
   2.1 Writing modules for VisionCLIPS ................. 5
      2.1.1 Structure of a module ..................... 5
      2.1.2 The DefineFunction() .................... 5
      2.1.3 An example module ....................... 5
   2.2 The shared module ................................ 5
      2.2.1 Purpose ..................................... 5
      2.2.2 The symbol table ......................... 7
      2.2.3 An example ............................... 7

3 Dynamic CLIPS ..................................... 9
   3.1 Basic concepts ................................ 9
      3.1.1 Linking C to CLIPS .................... 9
      3.1.2 Dynamic Linking ......................... 9
      3.1.3 Dld and CLIPS ............................ 9
      3.1.4 Advantages and Disadvantages ............ 9
   3.2 Modules for Dynamic CLIPS ...................... 10
      3.2.1 Structure of a module ................... 10
      3.2.2 The link-instructions .................... 10
      3.2.3 The DefineFunction() .................... 10
      3.2.4 An example module ....................... 11
   3.3 Using Dynamic CLIPS ............................ 11
      3.3.1 Linking modules .......................... 11
      3.3.2 New functions ............................ 11
   3.4 Internal structure ................................ 13
      3.4.1 The interfacing of Dld and CLIPS ......... 13
      3.4.2 Linking without DefineFunction() ......... 13
      3.4.3 Unlinking modules ....................... 14

4 List of Functions ................................ 14
   4.1 Graphic Functions ............................... 14
   4.2 Image Functions ................................ 18
   4.3 Miscellaneous Functions ....................... 19
   4.4 Corner Functions ................................ 34
   4.5 Contour Functions ............................... 42
   4.6 Calibration Functions ......................... 43
   4.7 Histogram Functions ............................ 44
   4.8 Line Functions ................................ 47
   4.9 Mathematical Functions ....................... 48
4.10 Matrix Functions .................................................. 49
4.11 Pyramid Functions .................................................... 52
4.12 Sequence Functions ................................................... 53
1 Introduction

1.1 Basic concepts

VisionCLIPS is an environment for developing vision-based applications. It consists of
CLIPS, an AI-shell with a lisp like syntax, as the command interpreter and a collection
of modules written in C that are linked to CLIPS.

This document describes the architecture of VisionCLIPS and serves as a users
guide how to use the existing modules and write new ones.

VisionCLIPS was developed at the LIFIA, Institut IMAG, Grenoble by the group
of Prof. Crowley.

1.2 CLIPS

CLIPS is an expert system shell developed by the Software Technology Branch of the
NASA. It supports rule based forward chaining using the Rete algorithm as well as
procedural and object oriented programming.

CLIPS enables the user to add Functions written in C to CLIPS and use them as
new CLIPS functions. This enables the user to write fast, optimized low-level routines
which then are integrated in an environment having the full advantage of an interpreted
high-level language.

1.3 Overview of existing functions

VisionCLIPS adds to the functionality of CLIPS a large number of functions for image-
processing, feature detection and scene reconstruction as well as functions for visual-
ization. Currently the following modules exist for VisionCLIPS:

<table>
<thead>
<tr>
<th>Contents</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic functions</td>
<td>graphic.o, line.o</td>
</tr>
<tr>
<td>Image handling</td>
<td>image.o, sequence.o</td>
</tr>
<tr>
<td>Corner detection and handling</td>
<td>corner.o</td>
</tr>
<tr>
<td>Contour extraction and handling</td>
<td>contour.o</td>
</tr>
<tr>
<td>Matching and reconstruction</td>
<td>match.o</td>
</tr>
<tr>
<td>Segment detection and handling</td>
<td>line.o, segment.o</td>
</tr>
<tr>
<td>Calibration</td>
<td>calib.o</td>
</tr>
<tr>
<td>Histogram functions</td>
<td>histogramme.o</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td>math.o, matrix.o</td>
</tr>
<tr>
<td>Resolution pyramid of an image</td>
<td>pyramid.o</td>
</tr>
</tbody>
</table>

1.4 Static and dynamic linking

When using VisionCLIPS the user has the choice to link his modules to CLIPS statically
or dynamically.

The advantages and disadvantages of dynamic linking for VisionCLIPS are dis-
cussed in the section DynamicCLIPS.
2 Modules for VisionCLIPS

2.1 Writing modules for VisionCLIPS

2.1.1 Structure of a module

The modules of CLIPS are a library from which the user chooses the part he needs for his application. A module for VisionCLIPS consists of a C source code file and a list of corresponding calls to DefineFunction(). A module contains two different types of procedures. Some procedures are connected directly to CLIPS using the DefineFunction() call while others are used only internally in the module. The procedures interfacing directly have to use special ways of passing argument and returning results. See the CLIPS users guide and the CLIPS reference manual for a detailed description. To ensure that all modules are compatible with each other, and modules can be either linked statically or dynamically certain rules should be adhered.

- All functions that are added to CLIPS using DefineFunction() should have a name beginning with clips_. The rest of the name should be the name of the clips-function witch dashes replaced by underscores (example: apple-eat would be clips_apple_eat). The reason for this is compatibility with DynamicCLIPS which is explained in section ??.

- The name of the function should consist of a first part describing the data type that is manipulated, a dash, and a second part describing the action. (example: apple-read, apple-write, apple-display but NOT read-apple)

- No function of a module should call a function of another module. Functions used by several modules should be stored in libraries.

- All functions except those added to CLIPS and all global variables should be static.

- The DefineFunction() calls for each module should be stored in a file with the same name as the module and the ending .def. This file should include prototypes for the corresponding functions to avoid compiler warnings.

2.1.2 The DefineFunction()

The DefineFunction() (Or for CLIPS version 6 the DefineFunction2()) calls to add the functions to CLIPS should be made in the UserFunction() of CLIPS or in the initialization function for the dynamic linker. Both simply include the file "modulename.def".

2.1.3 An example module

Figure 1 shows a short prototype model. For details on interfacing with CLIPS see the CLIPS users guide and programmers manual. See Figure 3 for a similar example for DynamicCLIPS.
------------- apple.c -------------
/*
 * Apple manipulation for clips
*/
#include <clips.h>

static int apples=5;

void clips_apple_set(void)
{
    apples = RtnFloat(1);
}

void clips_apple_display(void)
{
    ShowApples(apples); /* This is in a library */
}
-------------EOF apple.c -------------
------------- apple.def -------------
{
    extern void clips_apple_set(void);
    DefineFunction2("apple-set","v",clips_apple_set,
        "clips_apple_set","if");
}
{
    extern void clips_apple_display(void);
    DefineFunction2("apple-display","v",clips_apple_display,
        "clips_apple_set","if");
}
-------------EOF apple.def -------------

Figure 1: Example for a static clips module
3 The global CLIPS library

3.1 Purpose

The global CLIPS library is a module, or rather a library that provides functions to other VisionCLIPS modules to handle pointers. When using a function in VisionCLIPS that receives a pointer as an argument, and this pointer is invalid (for example 0) clips will probably crash.

The global CLIPS library provides a possibility to shield the underlying pointers from the user completely and use symbols instead. The shared module keeps a list of pointers and symbols and manages this list.

3.2 The symbol table

The symbol table is a double chained list of symbol nodes. Each node contains the symbol as a string, the corresponding pointer to an object and the type of the object.

CLIPS functions that want to use a certain object type call a function with the symbolic name of the object as the argument. They then get a pointer to the object (or NULL or -1 for invalid BARYs) as the return value. Currently the global CLIPS library supports images, corners, chains, segment lists and 3D world data. This list will keep expanding. Functions for new object types can be quickly derived from the existing ones.

`libglobalclips.a` provides functions storing, deleting, creating and recalling entries. If an error occurs it will be immediately printed to `stdout`. When `globalclips.c` is compiled it is transformed into a library called `libglobalclips.a`. The four functions provided for each object type are the following (with TYPE replaced by the corresponding object type):

- **TYPE *get_TYPE(char *name)**
  Searches the list for an object of this type with the given name. Returns a pointer to the object if successful, NULL (-1 for images) otherwise.

- **int put_TYPE(char *name, TYPE *obj)**
  Stores the object pointer and the name in the symbol table. Should be called when the object is created. Returns TRUE if successful, FALSE otherwise.

- **char *symb_new_TYPE(TYPE *object)**
  Creates a name for an object of the given type and adds it to the symbol table. This should be called if an object is created and the name doesn’t matter. Object names will typically be “newTYPEn” with n being a number starting with 0. Returns the name of the object in the symbol table.

- **int remove_TYPE(char *name)**
  Removes this entry from the symbol table. Should be called when an object is deleted. Returns TRUE if successful, FALSE otherwise.

3.3 An example

Figure 2 shows a simple example for using the global CLIPS library. If the proper functions for apples do not already exist they will have to be made from the existing
functions. Given an image pointer the function uses \texttt{get\_image} to get the image pointer. Afterwards a new symbol for an apple is requested and the apple pointer is stored using \texttt{new\_apple\_symb}.

```c
void clips_apple_detect(void)
{
    char fact[512];

    BARY *image;
    APPLE *apple;
    char *image_name;

    image_name = RtnLexeme(1); /* get the name */

    image = get_image(image_name);
    if( get_image(image_name) != (BARY *) -1 ) return;

    apple = DetectApple(image); /* Library */

    sprintf( fact,"apple %s in image %s",
        symb_new_apple(apple),image_name);
    MyAssert(fact);
}
```

Figure 2: Example for calling the global CLIPS library

4 DynamicCLIPS

4.1 Basic concepts

4.1.1 Linking C to CLIPS

The normal way of adding functions to CLIPS is writing them into a separate file, compiling it, linking it to the main CLIPS The functions are then added to CLIPS by calls to \texttt{DefineFunction()} in the main module. This method has several disadvantages.

- Each change in a module requires a relinking of the complete VisionCLIPS making test-compile cycles long.
- When adding or removing modules the clips source has to be changed.
- When adding modules to VisionCLIPS the user has to find out which libraries are now needed to link VisionCLIPS successfully.
The ideal solution would be an environment in which the user can specify the modules he wants to include into CLIPS while CLIPS is already running. This can be accomplished by dynamic linking.

4.1.2 Dynamic Linking

When linking executables the linker takes the separate object modules generated by the compiler. He resolves references between modules and from modules to libraries and merges the modules and the necessary library functions into one executable.

Dynamic linking enables the programme to perform some of these steps while it is already running. When the program is generated the static linker takes only a minimal necessary subset of the program, merges it with the dynamic-link library and generates an executable.

This executable has the possibility to link object modules while it is running. The dynamic linker will resolve references from the new modules to the executable and will provide the executable with means to call functions in the module. It further offers functions to unlink object modules and to detect if a function is executable.

4.1.3 Dld and CLIPS

The dynamic linker chosen for DynamicCLIPS is Dld from the Free Software Foundation. Dld is based on ld, the linker of the GNU-C package and the source code is available under the GPL. It is available on a large number of platforms. In Dld only minor changes had to be made to adapt it to CLIPS.

DynamicCLIPS provides the user with a new CLIPS instructions to link and un-link modules. After a module is linked DynamicCLIPS call an initialization function in the module which links the needed library and makes the necessary calls to DefineFunction().

4.1.4 Advantages and Disadvantages

Although DynamicCLIPS will normally be easier to use as static CLIPS it has some disadvantages. The advantages of DynamicCLIPS are:

- Easy to configure for a specific need. No recompiling/relinking if needed modules change.
- Short development cycles as DynamicCLIPS has not no be re-linked or even stopped to change a recompiled module.
- Information about necessary libraries contained in the module.
- Different versions of a module can be used in parallel.
- Easier to provide a central library of modules that can be updated constantly.

The disadvantages are:

- Debugging is harder as no source level debugger currently supports dynamic linking.
• Linking modules to DynamicCLIPS takes some time. When VisionCLIPS has to be restarted frequently (e.g. crashes) this might be a drawback.

• If all modules are included slightly higher memory consumption.

It should be noted that a properly written module for DynamicCLIPS can either be linked dynamically or statically.

4.2 Modules for DynamicCLIPS

4.2.1 Structure of a module

The structure of a DynamicCLIPS module is the same as described in 2.1.1 with two differences. An include file called dclipsmod.h has to included at the start of each module and an initialization function has to be provided.

The initialization function is called after the module has been linked. It contains the instructions which libraries have to be linked and the calls to DefineFunction().

4.2.2 The link-instructions

If a module needs further libraries it has to make sure that these are linked to the module. This is done by calling the did_link() routine of Dld. did_link() needs the complete path of the library and this path will differ on different systems. If the path was specified directly in each module, each module would have be changed if a library was moved. To avoid this the include file dclipsmod.h contains macros which are used for the calls to did_link(). This way only dclipsmod.h has to be changed.

The macros have the form DCL_LIBNAME where LIBNAME is the name of the library without leading “lib” and trailing “.a”.

4.2.3 The DefineFunction()

After the linking instructions the calls to DefineFunction() (or for CLIPS version 6 DefineFunction2()) have to be made. This should be done by including the appropriate NAME.def file. In the file dclipsmod.h a macro called DefineFunction() is defined that redirects the call to DefineFunction() to a slightly different function. This function behaves identically to DefineFunction() except that it performs a check if all subroutines that are called by a function have been linked. If some are missing and therefore the function is not safely executable an error is displayed and the function is not added to clips.

4.2.4 An example module

Figure 3 shows a short prototype model. For details on inter-facing with CLIPS see the CLIPS users guide and programmers manual. See Figure 1 for a similar example for CLIPS. If the macros for DCLAPPLE and DCLAPPLETREE do not already exist they have to be created in dclipsmod.h.
# --------------- apple.c ---------------

/*
 * Apple manipulation for clips
 */

#include <clips.h>
#include <dclslotsmod.h> /* In here the macros for dynamic
                             linking are defined. */

static int apples=5;

void clips_apple_set(void)
{
    apples = RtnFloat(1);
}

void clips_apple_display(void)
{
    ShowApples(apples); /* This is in a library */
}

void InitMoudle_apple(void)
{
    DCL_APPLE;       /* The famous apple library */
    DCL_APPLETREE;  /* This is needed by the apple library */
    DCL_LIBC;       /* Needed by the libraries */

    #include "apple.def"
}

# --------------- apple.def ---------------

{ extern void clips_apple_set(void);
    DefineFunction2("apple-set",'v',clips_apple_set,
            "clips_apple_set","iif");
}

{ extern void clips_apple_display(void);
    DefineFunction2("apple-display",'v',clips_apple_display,
            "clips_apple_set","iif");
}

# --------------- dclslotsmod.h ---------------

#define DCL_APPLE dld("/usr/lib/Apples/libapple.a");
#define DCL_APPLETREE dld("/usr/lib/Apples/libappletree.a");

Figure 3: Example of a dynamic clips module
4.3 Using DynamicCLIPS

4.3.1 Linking modules

Before using a dynamic module to DynamicCLIPS it has to be compiled. All libraries the module needs have to specified by the initialization function. If the module wants to call functions from the main executable this must contain the symbol hunk, that is it must be compiled with the -g option and may not be stripped using strip.

Clips programs that use the new functions have to be loaded after the modules containing the functions have been linked. Otherwise CLIPS may complain about unknown symbols.

4.3.2 New functions

---

**LINK-OBJ**

**Description:**

Links an object module to DynamicCLIPS, resolves references from the external module to CLIPS and initializes the module. The module will be searched in the current directory and the directory specified in the environment variable DCLIPS PATH.

**Syntax:**

```
(link-obj <filename>)
```

**Input:**

- `<filename>` The name of the module

---

**UNLINK-OBJ**

**Description:**

Unlinks an object module from DynamicCLIPS. The module will be searched in the current directory and the directory specified in the environment variable DCLIPS PATH. Unlinking of functions is only successful if their names follow the rules defined in section 2.1.1.

**Syntax:**

```
(unlink-obj <filename>)
```

**Input:**

- `<filename>` The name of the module

---

**DEBUG-LINK**

**Description:**
Enables detailed debugging output. This will display information which functions are added to CLIPS, which libraries are linked and how many references are still undefined after each step.

Syntax:

(debug-link)

4.4 Internal structure

This chapter describes some of the internal features of DynamicCLIPS. It contains no information that is necessary to use DynamicCLIPS and is intended for people that want to modify DynamicCLIPS itself.

4.4.1 The interfacing of Dld and CLIPS

When link-obj is called DynamicCLIPS calls the Dld function did_link(). This function loads the object file, resolves references and generates an internal table of functions of this module. The function in Dld that generates these entries calls a function in dynamic.cc that check each function if it is an InitModule... function or an automatic clips function (see section 3.4.2. The first is remembered, the second are stored in a list. After did_link() returns InitModule... is called (probably again causing the calls to did_link()). Finally the list of automatic functions is added using DefineFunction().

A part of the interface code is taken from Ravi, an expert-system shell that features dynamic loading developed at the LIFIA. The help of the developers of Ravi, especially of Bruno Zoppi is gratefully acknowledged.

4.4.2 Linking without DefineFunction()

It is possible to add functions to clips from a Module that does not contain an InitModule... As this method has several disadvantages over the previously described method it is NOT recommended to do this.

Functions that shall be added to CLIPS automatically start with cauto_ followed by the return type as in DefineFunction() and an underscore. To add the function test-it to CLIPS its name would have to be:

void cauto_v_test_it(void)

The problems of this method are that DefineFunction2() can not be used, that no linking instructions can be specified and that compatibility with static CLIPS is worse.

4.4.3 Unlinking modules

When unlinking functions VisionCLIPS calls didunlink_by_file() with the given filename and a parameter specifying a hard unlink as the argument. For details what a hard link means see the Dld users guide. The hard un-link is necessary because of the reference to the CLIPS functions in the InitModule function of the module. The hard un-link means on the other hand that DynamicCLIPS does not check if a function
is used by another module. Therefore a module that uses a function from a different module might crash if the other module is unlinked. Calls between modules will make unlinking dangerous.

As clips has no information how the name of the CLIPS function is it can only guess it from the name of the underlying C function. If the rules for naming clips functions as described in section 2.1.1 are not followed unlink is not able to un-define the CLIPS function correctly. If it does guess the name of the function this will be linked to a C dummy function printing an error message.

5 List of Functions

5.1 Graphic Functions

uses the xdraw functions

---

**WINDOW-CREATE**

**Description:**

Creates a window

**Syntax:**

```
(window-create [ <index> [ <name> [ <width> <height> <x-pos> <y-pos> ]]]]
```

**Input:**

- `<index>`: index used for the created window, default value 1
- `<name>`: name of the window to create, default name window 1 or window index if there is an index
- `<width>`: the width of the window in pixels, default-value is 512
- `<height>`: the height of the window in pixels, the default-value is 512
- `<x-pos>`: the coordinates for the upper left corner of the window.
- `<y-pos>`: The default-value is (0,0)

**Asserted fact:**

```
window <title> <width> <height> <x> <y> <index>
```

---

**WINDOW-CLOSE**

**Description:**

Closes the current window

**Syntax:**

```
(window-close)
```
Asserted fact:

    window active <index>

WINDOW-CHANGE

Description:

Changes the current window

Syntax:

    (window-change <index>)

Input:

    <index>  The index of the window to change to

WINDOW-CLEAR

Description:

Clears the current window

Syntax:

    (window-clear)

LINE

Description:

Draws a line between two points in the current window

Syntax:

    (line <x1> <y1> <x2> <y2> [ <thickness> ])

Input:

    <x1> <y1>  the coordinates for the beginning of the line
    <x2> <y2>  the coordinates for the end of the line
    <thickness>  the width of the line in pixels, the default level is 1

PLOT

Description:

Plots a point in the current window

Syntax:
(plot <x1> <y1>)

Input:

<x1> <y1>  the coordinates of the point

CROSS

Description:

Draws a + cross in the current window

Syntax:

(cross <x> <y> [ <thickness> [ <length> ] ])

Input:

<x> <y>  the coordinates for the center of the cross
<thickness>  the width of the lines in pixels, the default level is 1
[length>  length in pixels from the center of the cross to the end,
the default value is 5

SCROSS

Description:

Draws an x cross in the current window

Syntax:

(scross <x> <y> [ <thickness> [ <length> ] ])

Input:

<x> <y>  the coordinates for the center of the cross
<thickness>  the width of the lines in pixels, the default level is 1
[length>  length in pixels from the center of the cross to the end,
the default value is 5

COLOR

Description:

Sets the color

Syntax:

(color <color-name>)

Input:

<color-name>  name of the color, possible colors: white, black, red,
green, blue, yellow, magenta, cyan
TEXT

Description:

Prints a text in the current window

Syntax:

(text <string> [ <x> <y>])

Input:

<string> text to be printed
<x> <y> coordinate of the lower left corner of the text, default value (1,10) that is the text appears in the upper left corner

Asserted fact:

cursor get <point.x> <point.y>

CURSOR

Description:

Gets the cursor coordinates

Syntax:

(cursor)

Asserted fact:

(cursor get <point.x> <point.y>)

5.2 Image Functions

IMAGE-READ

Description:

reads an image

Syntax:

(image-read <filename>)

Input:

<filename> file in which the image is kept

Output:
<symb-name> of the read image

**Asserted fact:**

(image <file-name> <iname>)

---

**IMAGE-WRITE**

**Description:**

writes an image

**Syntax:**

(image-write <imagename> <filename> )

**Input:**

- `<imagename>` pointer which points to the image
- `<filename>` file in which the image will be stored

---

**IMAGE-DISPLAY**

**Description:**

displays an image in the current window

**Syntax:**

(image-display <imagepointer> [x y])

**Input:**

- `<imagepointer>` pointer to the image that will be displayed
- `<x>` default value 0
- `<y>` default value 0

---

**IMAGE-CREATE**

**Description:**

creates an image

**Syntax:**

(image-create [size [grayvalue]])

**Input:**

- `<size>` size of the image created, size = size
- `<grayvalue>` gray level of the created image, the default value is 0, that is a black image

**Asserted fact:**

(image of size <size> created <symb-new-image>)
5.3 Miscellaneous Functions

operations applied to the image

IMAGE-CONTRAST

Description:

stretches the gray values of an image between 0 and 255

Syntax:

(image-contrast <img-id> [ <min> <max> [ <inverse> ]])

Input:

<img-id> the reference of the image (usually its pointer)
<min> The minimum (default 255) and
<max> the maximum default 0) values used for the stretching
<inverse> used for a permutation of black and white pixels, de-

fault 0

Asserted fact:

(image contrasted <img-id> <img-id-computed>)

IMAGE-DIFF

Description:

Computes the difference between two images

Syntax:

(image-diff <img-id1> <img-id2> )

Input:

<img-id1> <img-id2> the references of the two images (usually their pointer)

Asserted fact:

(image difference <img-id1> <img-id2> <img-id-computed>)

IMAGE-EXTRACT

Description:

extracts a rectangular part of an image

Syntax:
Input:

- `<img-id>`: the reference of the image (usually its pointer)
- `<x0> <y0>`: the coordinates of the upper left
- `<x1> <y1>`: the lower right corners of the rectangular part
- `<border>`: if `<border>` equals to 1, these parameters are used
- `<border-value>`: to mark the border of this rectangular part with a line of gray level `<border-value>`

Asserted fact:

(image extracted `<img-id1>` `<img-id-computed>`)
Asserted fact:

(image minmax <img-id> max <max> at <x-max> <y-max> min <min> at <x-min> <y-min> )

IMAGE-MAGNITUDE

Description:
Computes the gradient magnitude of an image

Syntax:

(image-magnitude <img-id>)

Input:
<img-id> the reference of the image (usually its pointer)

Asserted fact:

(image magnitude <img-id> <img-id-computed>)

IMAGE-MAGNITUDE1

Description:
filtering with −101

Syntax:

(image-magnitude1 <img-id>)

Input:
<img-id> the reference of the image (usually its pointer)

Asserted fact:

(image magnitude1 <img-id> <img-id-computed>)

IMAGE-MAGNITUDE2

Description:
filtering with −11 in diagonal direction

Syntax:

(image-magnitude2 <img-id>)

Input:
<img-id> the reference of the image (usually its pointer)
**Asserted fact:**

(image magnitude2 <img-id> <img-id-computed>)

---

**IMAGE-MAGNITUDE3**

**Description:**

filtering with −101 in diagonal direction

**Syntax:**

(image-magnitude3 <img-id>)

**Input:**

<img-id> the reference of the image (usually its pointer)

**Asserted fact:**

(image magnitude3 <img-id> <img-id-computed>)

---

**IMAGE-LAPLACIAN**

**Description:**

Computes the second derivative (laplacian) of an image. The values of the laplacian are multiplied by 8, to obtain a more powerful laplacian image

**Syntax:**

(image-laplacian <img-id>)

**Input:**

<img-id> the reference of the image (usually its pointer)

**Asserted fact:**

(image laplacian <img-id> <img-id-computed>)

---

**IMAGE-NORMALIZED-LAPLACIAN**

**Description:**

Computes the second derivative (laplacian) of an image

**Syntax:**

(image-normalized-laplacian <img-id>)

**Input:**

<img-id> the reference of the image (usually its pointer)
 Asserted fact:

(image normalized laplacian <img-id> <img-id-computed>)

---

**IMAGE-EXPAND**

**Description:**

Expands the size of an image

**Syntax:**

(image-expand <img-id> [ <scale> ] )

**Input:**

- `<img-id>`: the reference of the image (usually its pointer)
- `<scale>`: (integer) the value of the scale expansion. The default value is such that the expanded image is the biggest image whose size is smaller than 512x512

**Asserted fact:**

(image expand <img-id> <img-id-computed>)

---

**IMAGE-BLACK-SUM**

**Description:**

Computes a kind of average image between two images: for each pixel location, the blackest pixel is used for the resulting image

**Syntax:**

(image-black-sum <img-id1> <img-id2> )

**Input:**

- `<img-id1>` `<img-id2>`: the references of the two images (usually their pointer)

**Asserted fact:**

(image black-sum <img-id1> <img-id2> <img-id-computed>)

---

**IMAGE-MAXLOCAL**

**Description:**

Computes the local maxima of an image. A local maximum is defined by its immediate neighbourhood (8 pixels)
Syntax:

(image-maxlocal <img-id> [ <x0> <y0> <x1> <y1> ]

Input:
- <img-id> the reference of the image (usually its pointer)
- <x0> <y0> the coordinates of the part of the image where we want to compute the local maxima.
- <x1> <y1> The default values are such that the computation is done on the whole image.

Asserted fact:

(image maxlocal <img-id> <img-id-computed>)

---

IMAGE-SMAXLOCAL

Description:

Computes the local maxima of an image. A local maximum is defined by its immediate neighbourhood (8 pixels)

Syntax:

(image-smaxlocal <img-id> [ <x0> <y0> <x1> <y1> ]

Input:
- <img-id> the reference of the image (usually its pointer)
- <x0> <y0> the coordinates of the part of the image where we want to compute the local maxima.
- <x1> <y1> The default values are such that the computation is done on the whole image.

Asserted fact:

(image maxlocal <img-id> <img-id-computed>)

---

IMAGE-THRESH

Description:

thresholds an image. Each pixel whose gray value is lower than the threshold is set to 0. The others pixels are left unchanged

Syntax:

(image-thresh <img-id> [ <threshold-value>[<dx> <dy>]]

Input:
- <img-id> the reference of the image (usually its pointer)
- <threshold-value> the value of the threshold. The default value is 128
- <dx> <dy> region of interest
Asserted fact:

(image threshed <img-id> <img-id-computed>)

DESCRIPTION

Description:
Computes the sizes of an image

Syntax:
(image-size <img-id> )

Input:
<img-id> the reference of the image (usually its pointer)

Asserted fact:
(image size <x-min> <x-max> <y-min> <y-max> )

IMAGE-SIZE-MIN-X

Description:
Computes the most left location of the pixels of an image. That is the minimum row index.

Syntax:
(image-size-min-x <img-id> )

Input:
<img-id> the reference of the image (usually its pointer)

Output:
the most left value of the pixels of this image

IMAGE-SIZE-MAX-X

Description:
Computes the most right location of the pixels of an image. That is the maximum row index.

Syntax:
(image-size-max-x <img-id> )

Input:
<img-id> the reference of the image (usually its pointer)

Output:
the most right value of the pixels of this image

---

**IMAGE-SIZE-MIN-Y**

**Description:**
Computes the upper most location of the pixels of an image. That is the minimum column index.

**Syntax:**
(image-size-min-y <img-id> )

**Input:**
<img-id> the reference of the image (usually its pointer)

**Output:**
the upper most value of the pixels of this image

---

**IMAGE-SIZE-MAX-Y**

**Description:**
Computes the lowest location of the pixels of an image. That is the maximum column index

**Syntax:**
(image-size-max-y <img-id> )

**Input:**
<img-id> the reference of the image (usually its pointer)

**Output:**
the lowest value of the pixels of this image

---

**IMAGE-SIZE-X**

**Description:**
Compute the size of the rows of an image

**Syntax:**
(image-size-x <img-id> )
Input:  
\(<\text{img-id}>\)  the reference of the image (usually its pointer)

Output:  
the length of the rows of this image

---

IMAGE-SIZE-Y

Description:  
Compute the size of the columns of an image

Syntax:  

\((\text{image-size-y} \ <\text{img-id}> \ )\)

Input:  
\(<\text{img-id}>\)  the reference of the image (usually its pointer)

Output:  
the length of the columns of this image

---

IMAGE-SEARCHBLACK

Description:  
computes the location of the gravity center as well as the number of points of the biggest black region of an image

Syntax:  

\((\text{image-searchblack} \ <\text{img-id}> \ )\)

Input:  
\(<\text{img-id}>\)  the reference of the image (usually its pointer)

Asserted fact:  

\((\text{bary searchblackened} \ <\text{img-id}> \ <\text{x-location}> \ <\text{y-location}> \ <\text{nb-points}> \ )\)

---

IMAGE-GET-PIXEL

Description:  
Gets the gray value of a pixel of an image

Syntax:  

\((\text{image-get-pixel} \ <\text{img-id}> \ <\text{x}> \ <\text{y}> \ )\)
**Input:**
- `<img-id>` the reference of the image (usually its pointer)
- `<x> <y>` the location of the pixel

**Output:**
- the gray value of this pixel

**IMAGE-SET-PIXEL**

**Description:**
Sets the gray value of a pixel of an image

**Syntax:**
```
(image-set-pixel <img-id> <x> <y> )
```

**Input:**
- `<img-id>` the reference of the image (usually its pointer)
- `<x> <y>` the location of the pixel

**IMAGE-SEUIL**

**Description:**
Marks all the pixels of the image whose gray value is between two values

**Syntax:**
```
(image-seuil <img-id> <g0> <g1> )
```

**Input:**
- `<img-id>` the reference of the image (usually its pointer)
- `<g0> <g1>` the two values

**IMAGE-ISO**

**Description:**
Marks all the pixels of an image whose gray values is equal to a value

**Syntax:**
```
(image-iso <img-id> <value> )
```

**Input:**
- `<img-id>` the reference of the image (usually its pointer)
- `<value>` the searched value
IMAGE-FREE

Description:

de-allocates the memory used for an image

Syntax:

(image-free <img-id> )

Input:

<img-id>  the reference of the image (usually its pointer)

Asserted fact:

(image free <img-id> )

IMAGE-G3

Description:

Computes a low pass filtered image of an image using the gaussian filter g3

Syntax:

(image-g3 <img-id> )

Input:

<img-id>  the reference of the image (usually its pointer)

Asserted fact:

(image g3 <img-id> <img-id-computed>)

IMAGE-G5

Description:

Computes a low pass filtered image of an image using the gaussian filter g5

Syntax:

(image-g5 <img-id> )

Input:

<img-id>  the reference of the image (usually its pointer)

 Asserted fact:

(image g5 <img-id> <img-id-computed>)
IMAGE-G7

Description:

Computes a low pass filtered image of an image using the gaussian filter g7

Syntax:

(image-g7 <img-id> )

Input:

<img-id>   the reference of the image (usually its pointer)

Asserted fact:

(image g7 <img-id> <img-id-computed>)

IMAGE-G13

Description:

Computes a low pass filtered image of an image using the gaussian filter g13

Syntax:

(image-g13 <img-id> )

Input:

<img-id>   the reference of the image (usually its pointer)

Asserted fact:

(image g13 <img-id> <img-id-computed>)

IMAGE-THIN

Description:

???

Syntax:

(image-thin <image-name>)

Input:

<image-name>   the reference of the image (usually its pointer)

Asserted fact:
(image thined `<img-name>` `<symb-new-image>`)

**IMAGE-ADAPT-THRESH**

**Description:**

???

**Syntax:**

(\texttt{image-adapt-thresh} `<image-name>` [\texttt{<size> <fact-sigma> <thresh>}]\texttt{)}

**Input:**

<table>
<thead>
<tr>
<th><code>&lt;image-name&gt;</code></th>
<th>the reference of the image (usually its pointer)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;size&gt;</code></td>
<td>???</td>
</tr>
<tr>
<td><code>&lt;fact-sigma&gt;</code></td>
<td>???</td>
</tr>
<tr>
<td><code>&lt;thresh&gt;</code></td>
<td>???</td>
</tr>
</tbody>
</table>

**Asserted fact:**

(\texttt{image adapted thresh} `<img-name>` `<symb-new-image>`)

**IMAGE-ADAPT-THRESH2**

**Description:**

???

**Syntax:**

(\texttt{image-adapt-thresh2} `<image-name>` [\texttt{<size> <fact-sigma> <thresh>}]\texttt{)}

**Input:**

<table>
<thead>
<tr>
<th><code>&lt;image-name&gt;</code></th>
<th>the reference of the image (usually its pointer)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;size&gt;</code></td>
<td>???, default 4</td>
</tr>
<tr>
<td><code>&lt;fact-sigma&gt;</code></td>
<td>???, default 1.5</td>
</tr>
<tr>
<td><code>&lt;thresh&gt;</code></td>
<td>???, default 20</td>
</tr>
</tbody>
</table>

**Asserted fact:**

(\texttt{image adapted thresh2} `<img-name>` `<symb-new-image>`)

**IMAGE-INFO-DIFF**

**Description:**
Syntax:

(image-info-diff <image-name1> <image-name2>)

Input:

<image-name1> the reference of the first image (usually its pointer)
<image-name2> the reference of the second image (usually its pointer)

Asserted fact:

(diff info <img-name1> <img-name2> max <max> moy <moyenne> ecart <ecart>)

IMAGE-MAGNITUDE4

Description:

Computes the gradient magnitude of an image (Canny/Dericke)

Syntax:

(image-magnitude4 <img-id>)

Input:

<img-id> the reference of the image (usually its pointer)

Asserted fact:

(image magnitude4 <img-id> <img-id-computed> high <uptresh> low <lowthresh>)

IMAGE-DIFF-ABSOLU

Description:

???

Syntax:

(image-diff-absolu <image-name1> <image-name2>)

Input:

<image-name1> the reference of the first image (usually its pointer)
<image-name2> the reference of the second image (usually its pointer)

Asserted fact:

(absolute image difference <img-name1> <img-name2> <symb-new-image>)
IMAGE-ADD-NOISE-UNIFORME

Description:

???

Syntax:

(image-add-noise-uniforme <image-name> [<scale>])

Input:

<image-name>  the reference of the image (usually its pointer)
<scale>       default 1

Asserted fact:

(image <img-name> uniforme noisy image <symb-new-image>)

IMAGE-ADD-NOISE-GAUSSIAN

Description:

???

Syntax:

(image-add-noise-gaussian <image-name> [<mean> <sigma>])

Input:

<image-name>  the reference of the image (usually its pointer)
<mean>        default 0
<sigma>       default 1

Asserted fact:

(image <img-name> gaussian noisy image <symb-new-image>)

5.4 Corner Functions

A list of corners has as first argument the number n of corners contained in the list, then follow the n corners

CORNER-DISPLAY

Description:

Displays a list of corners
Syntax:

```
(corner-display <cornerlist>)
```

**Input:**

- `<cornerlist>`: the list of corners to be displayed

---

**CORNER-NBR**

**Description:**

Returns the number of corners of a list

**Syntax:**

```
(corner-nbr <cornerlist>)
```

**Input:**

- `<cornerlist>`: the list of corners for which we want to know the number of corners

**Output:**

- `<nbr>`, number of corners

---

**CORNER-CREATE**

**Description:**

Creates a list of corners

**Syntax:**

```
(corner-create <number>)
```

**Input:**

- `<number>`: the number of corners in the created list

**Assertion fact:**

- list of corners created `<corners>`

---

**CORNER-SET**

**Description:**

Set a corners in a list

**Syntax:**

```
(corner-set <cornerlist><i><x><y>)
```

**Input:**
<cornerlist>  the list of corners in which the corner is set
<i>        the number of the corner to be set
<x>,<y>    the x and y values of the corner

CORNER-NTH-X

Description:

Returns the x-value of a corner in a list

Syntax:

(corner-nth-x <cornerlist><i>)

Input:
<cornerlist>  the list of corners
<i>         the number of the corner in the list

Output:
<x> coordinate of the ith corner

CORNER-NTH-Y

Description:

Returns the y-value of a corner in a list

Syntax:

(corner-nth-y <cornerlist><i>)

Input:
<cornerlist>  the list of corners
<i>         the number of the corner in the list

Output:
<y> coordinate of the ith corner

CORNER-READ

Description:

Reads a list of corners to a file

Syntax:

(corner-read <filename>)

Input:
<filename>  the file from which the corners are read

**Asserted fact:**

(list of corners <file-name> <corners>)

---

**CORNER-WRITE**

**Description:**

Writes a list of corners to a file

**Syntax:**

(cornert-write <cornerlist> <filename> )

**Input:**

<cornerlist>  the list of corners
<filename>  file to which we want to write the list of corners

---

**CORNER-EXTRACT**

**Description:**

extracts corners from a list of corners

**Syntax:**

(cornert-extract <cornerlist> <x1> <y1> <x2> <y2> )

**Input:**

<cornerlist>  the list of corners from which we extract
<x1> <y1>  coordinates for the beginning of the extraction
<x2> <y2>  coordinates for the end of the extraction

**Asserted fact:**

(corners extracts <cornersname> <corners-out>)

---

**CORNER-EXPAND**

**Description:**

expands a list of corners

**Syntax:**

(cornert-expand <cornerlist> <factor> )

**Input:**
<cornerliste>  the list of corners from which we expand
<factor>  the expansion factor

Asserted fact:

(corners expanded <cornersname> <corners-out>)

CORNER-DIFF

Description:

Computes the difference between two lists of corners

Syntax:

(corner-diff <cornerlist1> <cornerlist2> [thresh])

Input:
<cornerlist1>  the first list of corners
<cornerlist2>  the second list of corners
<thresh>  metric distance for which two corners are the same,
          default value 0

Asserted fact:

(corn difference <name1> <name2> <result>)

CORNER-INTERSECT

Description:

Computes the intersection between two lists of corners

Syntax:

(corner-intersect <cornerlist1> <cornerlist2> [<thresh>])

Input:
<cornerliste1>  the first list of corners
<cornerliste2>  the second list of corners
<thresh>  metric distance for which two corners are the same,
          default value 0

Output:

<nbr>, number of corners, which found in both lists

Asserted fact:

(corn intersection <name1> <name2> <result>)
CORNER-ZURICH

Description:
Computes corners of an image using function zurich

Syntax:
(corner-zurich <imagepointer> [ <sigma> <sweep> <thresh> <nori> <orisel> ])

Input:
<imagepointer> pointer to the image for which the corners are computed
<sigma> sigma of the gaussian envelope, default value 5.0
<sweep> sweep parameter of the egabor filter, default value 1.2
<thresh> thresh, default value 10.0
<nori> number of orientations, default value 6
<orisel> orientation selectivity, default value 4.0

Asserted fact:
(zurich image <image-name> corners <corners-out>)

CORNER-ZURICH-RAW

Description:
Computes corners of an image using function zurich

Syntax:
(corner-zurich-raw <imagename> [ <sigma> <sweep> <thresh> <nori> <orisel> ])

Input:
<imagepointer> pointer to the image for which the corners are computed
<sigma> sigma of the gaussian envelope, default value 5.0
<sweep> sweep parameter of the egabor filter, default value 1.2
<thresh> thresh, default value 10.0
<nori> number of orientations, default value 6
<orisel> orientation selectivity, default value 4.0

Asserted fact:
(zurich image <image-name> image-out<bary-img-out> corners <corners-out>)
CORNER-ZURICH1

Description:
Computes corners of an image using function zurich1

Syntax:
(cornert-zurich1 <imagepointer> [ <sigma> <sweep> <thresh>
<nori> <orisel>])

Input:
<imagepointer> pointer to the image for which the corners are computed
<sigma> sigma of the gaussian envelope, default value 5.0
<sweep> sweep parameter of the egabor filter, default value 1.2
<thresh> thresh, default value 10.0
<nori> number of orientations, default value 6
<orisel> orientation selectivity, default value 4.0

Asserted fact:
(image <image-name> kptimage <bary-img-out>filtered images <filtered-
images> corners <corners-out>)

CORNER-DROID

Description:
Computes corners of an image using function droid

Syntax:
(cornert-droid <imagpointer> [ <sigma> <relminimum>
<scalefactor>])

Input:
<imagpointer> pointer to the image for which the corners are computed
<sigma> sigma, default value 2.5
<relminimum> relative minimum, default value 0.01
<scalefactor> scalefactor, default value 0.06

Asserted fact:
(droid image < img-name> corners <corner>)
Description:

Computes corners of an image using function beaudet

Syntax:

\[(\text{corner-beaudet} \ <\text{imagpointer}> \ [<\text{thresh}>])\]

Input:

\(<\text{imagpointer}>\)  poitner to the image for which the corners are computed
\(<\text{thresh}>\)  thresh, default value 0.6

Asserted fact:

beaudet image <image-name> corners <corner>

Description:

??? Computes corners of an image using function foerstner

Syntax:

\[(\text{corner-foerstner} \ <\text{imagpointer}>\)]

Input:

\(<\text{imagpointer}>\)  poitner to the image for which the corners are computed

Asserted fact:

foerstner image <image-name> corners <corner>

Description:

???

Syntax:

\[(\text{corner-regression} \ <\text{imagpointer}>\)]

Input:
<imagepointer> pointer to the image for which the corners are computed

Asserted fact:

linear regression points <corners-name> <a> <b> <c> residual <sr2>)

CORNER-DETECT-CLOUD

Description:

Computes a cloud image from a corner image

Syntax:

(corner-detect-cloud <cornerimage> [ <dist> <nbrpoints>])

Input:

<cornerimage> image of the corners
<dist> size of a cloud, default value 4
<nbrpoints> number of points which have to be in a cloud, default value 1

Asserted fact:

(image <image-name> cloud corners jcorners-out)

CORNER-PRECISE

Description:

Computes the precise corner

Syntax:

(corner-precise <image> <x> <y> [<dx> <dy> <type>]

Input:

<image> image for which the corners were computed and which is used to precise them
<x>,<y> x and y coordinates of the corner
<dx>,<dy> dx and dy are size of the approximation area in x and y direction, default values are 20, 20
<type> type of the corner, default value is c

Asserted fact:

(corner <corner-in.x> <corner-in.y> precise <corner-out.x> <corner-out.y> theta <theta> beta <beta> [beta2 <beta2> error <error>])
5.5 Contour Functions

CONTOUR-EXTRACT

Description:

extracts point-chains out of a gradient-image

Syntax:

(contour-extract <imagename> [<up-thresh> <down-thresh> <chainlength> [<max-nbr-chains>]])

Input:

<imagename>       Pointer of the gradient-image
<up-thresh>       maximum accepted distance from a point belonging to
the chain, default value is 1
<down-thresh>     minimum accepted distance from a point belonging to
the chain, default value is 1
<chainlength>     minimum length of a saved chain, default value is 5
<max-nbr-chains>  maximum-number of saved chains, default value is 10000

Asserted fact:

(image <image-name> chains in <symb-new-chains>)

CONTOUR-IMAGE

Description:

shows in a new window the chains on the screen (creates a new black/white image)

Syntax:

(contour-image <chain-name> [<dx> <dy>])

Input:

<chain-name>       pointer to the chainlist
(dx)              width of the new image, default value is 512
(dy)              height of the new image, default value is 512

Asserted fact:

(chain <chain-name> contour image <symb-new-image>)

CONTOUR-DISPLAY

Description:
shows in the original image the chains on the screen.

Syntax:
(contour-display <chain-name>)

Input: <chain-name>  Pointer to displayed chain

5.6 Calibration Functions

CALIB-RECONSTRUCTION

Description:
calculates the 3-D coordinates of a point (uses line/plane intersection)

Syntax:
(calib-reconstruction <matrixL-name> <matrixR-name> <PL.i> <PL.j> <PR.i> <PR.j>)

Input:
<matrixL-name>  left image of the scene
<matrixR-name>  right image of the scene
<PL.i>  x-coordinate of the point in the left image
<PL.j>  y-coordinate of the point in the left image
<PR.i>  x-coordinate of the point in the right image
<PR.j>  y-coordinate of the point in the right image

Asserted fact:
(point reconstruct <result.x> <result.y> <result.z>)

CALIB-RECONSTRUCTION-LINES

Description:
calculates the 3-D coordinates of a point (uses line/line intersection)

Syntax:
(calib-reconstruction-lines <matrixL-name> <matrixR-name> <PL.i> <PL.j> <PR.i> <PR.j>)

Input:
<matrixL-name>  left image of the scene
<matrixR-name>  right image of the scene
<PL.i>  x-coordinate of the point in the left image
<PL.j>  y-coordinate of the point in the left image
<PR.i>  x-coordinate of the point in the right image
<PR.j>  y-coordinate of the point in the right image
Output:
(x, y, z), the coordinates of the point in the 3-D scene

Asserted fact:
(point reconstruct <result.x> <result.y> <result.z>)

5.7 Histogram Functions

IMAGE-HISTOGRAMME

Description:
calculates the histogramme of an image

Syntax:
(image-histogramme <image-name>)

Input:
<image-name> Pointer to the image, of which the histogramme should be calculated

Asserted fact:
(histogramme from <image-name> <histogramme>)

HISTOGRAMME-DISPLAY

Description:
shows the histogramme on the screen

Syntax:
(histogramme-display <histo-name>)

Input:
<histo-name> Pointer to the histogramme

HISTOGRAMME-DISPLAY-LOG

Description:
shows the histogramme after using the logarithm-function

Syntax:
(histogramme-display-log <histo-name>)

Input:
<histo-name>  pointer to the histogramme

HISTOGRAMME-DISPLAY2

Description:

???

Syntax:

(histogramme-display2 <histo-name>)

Input:

<histo-name>  Pointer to the histogramme

HISTOGRAMME-DISPLAY2-LOG

Description:

???

Syntax:

(histogramme-display2-log <histo-name>)

Input:

<histo-name>  Pointer to the histogramme

HISTOGRAMME-GET

Description:

gets the frequency-value of a histogramme for a given grayvalue

Syntax:

(histogramme-get [<histo-name> [<int>]])

Input:

<histo-name>  Pointer to the histogramme
<int>        the given grayvalue

Output:

returns the value of the histogram

HISTOGRAMME-SET

Description:
sets frequency-value for a graylevel in a histogramme

Syntax:

\[(\text{histogramme-set} [<\text{histo-name}> [<i> <j>]])\]

Input:

\(<\text{histo-name}>\) Pointer to the histogramme
\(<i>\) concerned grayvalue
\(<j>\) new value of the frequency

---

**HISTOGRAMME-ADD**

**Description:**

adds two histogrammes

**Syntax:**

\[(\text{histogramme-add} <\text{histo-name1}> <\text{histo-name2}>)\]

**Input:**

\(<\text{histo-name1}>\) Pointer to the first histogramme
\(<\text{histo-name2}>\) Pointer to the second histogramme

**Asserted fact:**

\[(\text{add histogrammes} <\text{histoname1}> <\text{histoname2}> <\text{histo-sum}>)\]

---

**HISTOGRAMME-WRITE**

**Description:**

saves a histogramme to a file

**Syntax:**

\[(\text{histogramme-write} <\text{histo-name}> <\text{file-name}>)\]

**Input:**

\(<\text{histo-name}>\) Pointer to the histogramme
\(<\text{file-name}>\) name of the file

---

**5.8 Line Functions**

**LINE-DISPLAY**

**Description:**

displays a line on the screen
Syntax:

(line-display [<theta> <rho>][<a> <b> <c>])

Input:

<theta>  angle between the line and the x-axe
<rho>    distance of the line to (0,0)
<a>      the line is determined with \( ax + by + c = 0 \)
<b>
<c>

---

LINE-PRECISE

Description:

looks for a segment (line) near the given coordinates

Syntax:

(line-precise <image-name> <x> <y> [<dx> <dy> <type> [<p1> <p2>]])

Input:

<image-name>  pointer to the concerned image
<x>           x-coordinate of the point
<y>           y-coordinate of the point
<dx>          position of region of interest
<dy>          position of region of interest
<type>        e, for edge
<p1>          initial estimation for theta
<p2>          initial estimation for rho

Asserted fact:

(edge <x> <y> precise <theta> <roh.new> error <error>)

---

LINE-CHECK

Description:

computes how much percent of an amount of points, lying in the near environment of a line

Syntax:

(line-check <image-name> <x0> <y0> <x1> <y1> <size> <corners-name>)

Input:
\begin{itemize}
  \item \texttt{<image-name>}: pointer to the considered image
  \item \texttt{<x0>}: x-coordinate of the beginning point of the line
  \item \texttt{<y0>}: y-coordinate of the beginning point of the line
  \item \texttt{<x1>}: x-coordinate of the ending point of the line
  \item \texttt{<y1>}: y-coordinate of the ending point of the line
  \item \texttt{<size>}: maximum distance of the line to the points
  \item \texttt{<corners-name>}: list of the points
\end{itemize}

Output:
\begin{itemize}
  \item \texttt{<percent>}
\end{itemize}

5.9 Mathematical Functions

\underline{\texttt{RAND}}

Description:
\begin{quote}
  calculates a random number
\end{quote}

Syntax:
\begin{quote}
  \texttt{(rand \textless seed\textgreater )}
\end{quote}

Input:
\begin{itemize}
  \item \texttt{<seed>}: start value for the calculation
\end{itemize}

Output:
\begin{itemize}
  \item returns a random number
\end{itemize}

\underline{\texttt{SRAND}}

Description:
\begin{quote}
  calculates a random number
\end{quote}

Syntax:
\begin{quote}
  \texttt{(srand \textless seed\textgreater )}
\end{quote}

Input:
\begin{itemize}
  \item \texttt{<seed>}: start value for the calculation
\end{itemize}

Output:
\begin{itemize}
  \item returns a random number
\end{itemize}
RANDOM

Description:
calculates a random number

Syntax:
(random <seed>)

Input: <seed> startvalue for the calculation

Output:
returns a random number

SRANDOM

Description:
calculates a random number

Syntax:
(srandom <seed>)

Input: <seed> startvalue for the calculation

Output:
returns a random number

5.10 Matrix Functions

MATRIX-ALLOC

Description:
allocates memory for a matrix with "n-rows" rows and "n-cols" cols

Syntax:
(matrix-alloc <n-row> <n-col>)

Input: <n-rows> number of the needed rows
      <n-cols> number of the needed cols

Asserted fact:
(matrix "memory" <result> <n-rows> <n-cols>)

MATRIX-ZERO

Description:
initializes a matrix with 0

Syntax:
(matrix-zero <matrix-id> <n-row> <n-col>)

Input:
<matrix-id> Pointer to matrix
<n-rows> number of the rows
<n-cols> number of the cols

MATRIX-READ

Description:
reads a matrix from a file

Syntax:
(matrix-read <matrix-name> <n-row> <n-col>)

Input:
<matrix-name> name of the file
<n-rows> number of the rows
<n-cols> number of the cols

Asserted fact:
matrix <matrix-name> <result> <n-row> <n-col>)

MATRIX-FREE

Description:
deallocates the memory of the matrix

Syntax:
(matrix-free <matrix-id> <n-row>)

Input:
<matrix-id> pointer to the matrix
<n-rows> number of the rows
MATRIX-MULT

**Description:**

multiplicates two matrices

**Syntax:**

```
(matrix-read <matrix1-id> <n1-row> <n1-col> <matrix2-id> <n2-row> <n2-col>)
```

**Input:**

- `<matrix1-id>` pointer to the first matrix
- `<n1-rows>` number of the rows of the first matrix
- `<n1-cols>` number of the cols of the first matrix
- `<matrix2-id>` pointer to the second matrix
- `<n2-rows>` number of the rows of the second matrix
- `<n2-cols>` number of the cols of the second matrix

**Asserted fact:**

```
(matrix "memory" <result> <n1-row> <n2-col> <matrix1-id> <matrix2-id>)
```

MATRIX-PRINT

**Description:**

shows a matrix on the screen

**Syntax:**

```
(matrix-print <matrix-id> <n-row> <n-col>)
```

**Input:**

- `<matrix-id>` pointer to the matrix
- `<n-rows>` number of the rows
- `<n-cols>` number of the cols

MATRIX-SET

**Description:**

sets a value of a matrix at the position (n-row, n-col)

**Syntax:**

```
(matrix-set <matrix-id> <n-row> <n-col> <value>)
```

**Input:**
\begin{verbatim}
<matrix-id>  pointer to the matrix
<n-rows>     x-coordinate of the new value
<n-cols>     y-coordinate of the new value
<value>      the new set value
\end{verbatim}

**MATRIX-GET**

**Description:**

gets a value of a matrix at the position (n-row, ncol)

**Syntax:**

\[ (\text{matrix-print} \ <\text{matrix-id}> \ <\text{n-row}> \ <\text{n-col}>) \]

**Input:**

- `<matrix-id>`  pointer to the matrix
- `<n-rows>`     number of the rows
- `<n-cols>`     number of the cols

**Output:**

returns the value

**5.11 Pyramid Functions**

**PYRAMID-COMPUTE**

**Description:**

Computes the pyramid of resolution of an image

**Syntax:**

\[ (\text{pyramid-compute} \ <\text{imagepointer}>) \]

**Input:**

- `<imagepointer>`  pointer to the image for which the pyramid is computed

**Asserted fact:**

\[ (\text{image} \ <\text{image-name}> \ \text{pyramid} \ <\text{pyr}>) \]
5.12 Sequence Functions

SEQUENCE-EXTRACT

Description:
Extracts the nth image in a list of images

Syntax:
(sequence-extract <sequencepointer> <level>)

Input:
<sequencepointer>  pointer to the list of images
<level>             the levelth image is extracted

Asserted fact:
(image <pyr-name> niveau <niveau> <pyr>)
Index

calib-reconstruction, 44
clips, 4
color, 17
count-display, 44
count-extract, 43
count-image, 44
counter-beaudet, 41
counter-create, 35
counter-detect-cloud, 42
counter-diff, 38
counter-display, 35
counter-droid, 41
counter-expand, 38
counter-extract, 37
counter-foerstner, 41
counter-intersect, 39
counter-nbr, 35
counter-nth-x, 36
counter-nth-y, 36
counter-precise, 43
counter-read, 37
counter-regression, 42
counter-set, 36
counter-write, 37
counter-zurich, 39
counter-zurich-raw, 40
counter-zurich1, 40
cross, 16
cursor, 18

dcl, 10, 11
def-link, 13
def-file, 5
define-function, 5

dkd, 9

fsf, 9

gget, 7
gget_image, 7

 histogramme-add, 48
 histogramme-display, 46
 histogramme-display-log, 46
 histogramme-display2, 46
 histogramme-display2-log, 47
 histogramme-get, 47
 histogramme-set, 47
 histogramme-write, 48

 image-adapt-thresh, 32
 image-adapt-thresh2, 32
 image-add-noise-gaussian, 34
 image-add-noise-uniforme, 34
 image-black-sum, 24
 image-contrast, 19
 image-create, 19
 image-diff, 20
 image-diff-absolute, 34
 image-display, 19
 image-expand, 24
 image-extract, 20
 image-free, 30
 image-g13, 31
 image-g3, 30
 image-g5, 31
 image-g7, 31
 image-get-pixel, 29
 image-histogramme, 45
 image-info, 21
 image-info-diff, 33
 image-iso, 30
 image-laplacian, 23
 image-magnitude, 21
 image-magnitude1, 22
 image-magnitude2, 22
 image-magnitude3, 22
 image-magnitude4, 33
 image-maxlocal, 24
 image-minmax, 21
 image-normalized-laplacian, 23
 image-read, 18
 image-searchblack, 28
 image-set-pixel, 29
 image-seuil, 29
 image-size, 26
 image-size-max-x, 26
 image-size-max-y, 27