0

VIS Interpretive Control Language (VISICL).

Kenneth M. Dawson, Paul Healy, and David Vernon.

> If a conceptual description is to be made, the machinery for making it ought to show itself in language. If a distinction cannot be made in language, it can not be made conceptually.
>
> *N.R. Hanson.*

10.1 Motivation for a vision language.

The Virtual Image System Interpretive Control Language (or VISICL for short) was originally conceived and designed as a means to effect an interface between an low-level vision environment (running *VIS à VIS*) and a high-level vision or cognitive environment, running on a remote workstation. Thus, it was designed as a type of quick-turnaround 'batch processing' interface which would allow other workstations to control *VIS à VIS*, setting up VISICL programs which were then executed by the VISICL interpreter in *VIS à VIS*. The results, of course, were then communicated back to the originating workstation. Having realized this original intention, VISICL was subsequently developed as an interactive tool, i.e. a programming environment, for *VIS à VIS*. It proved extremely useful in this rôle for investigating complex visual processes involving several image transfers, ranging from tasks such as the computation of depth from camera motion to 3-D object recognition. Moreover, VISICL has now been extended so that it supports parallel processing, allowing any computer vision system to control instantiations of *VIS à VIS* (through VISICL in exactly

the same manner as that in which high-level workstations originally 'programmed' *VIS à VIS*). This facility is employed within VISICL itself in order to allow multiple copies of *VIS à VIS* to be executed in parallel (in a multi-processor MIMD environment), facilitating the execution of complex operations in a structured and efficient fashion.

10.2 VISICL - An Introduction.

First, and foremost, VISICL is a programming language, providing a number of control structures (e.g. *IF-THEN-ENDIF, WHILE-DO-ENDWHILE*) and a number of variable types (e.g. integers). However, it is a language designed to meet the very specific needs of visual processing and analysis, and to that end it provides a large number of primitives which perform a diverse range of visual functions (from image capture to more complex operations such as the computation of depth using camera motion). VISICL is also an interpretive language, in that the VISICL source text is translated into the required function calls and flow control primitives as it is being executed (i.e. at runtime), rather than having to be compiled prior to execution. As such, it provides a flexible and reasonably efficient environment in which complex visual techniques may be developed, through the combination of functional primitives (i.e. as a VISICL program or procedure) and through experimentation with new primitives.

This chapter is intended to introduce both the concept and the practicality of VISICL, using sample code drawn from various VISICL programs which, hopefully, will give an intuitive feel for the language. A formal definition of the syntax language is given in section 10.7, although the semantics have not been explicitly defined. This would require a complete programming manual and is outside the scope of this chapter. In any case, the semantics of most of the constructs and functional primitives, if not their parameter lists, can be inferred from context.

10.3 Image Processing.

The basis of any computer vision system is image processing and this section introduces the mechanisms by which image processing facilities may be invoked from within VISICL. The example program which follows is a complete program which computes the intensity discontinuities (see chapter 3) present in an intensity image stored on disk as a virtual framestore, and which displays the results through a framestore. The operation is equivalent to the computation of a particular spatial channel (see chapter 6) and, in fact, the program listed is merely part of a larger program which computes the raw primal sketch.

There is a large amount of information about VISICL implicitly encoded in the listing and it is worth reading it reasonably closely, bearing in mind a few points:

- An image is defined within the pyramid structure by the pyramid number and the level of the image within that pyramid (i.e. two integers). For example

the first *add_image* statement generates a mapping to a PCPLUS framestore as pyramid 1, level 2. The first *transfer_image* statement listed transfers pyramid 1, level 4 to pyramid $index + 2$, level 4 (i.e. loads the stored intensity image into an intensity array image).

- Most image processing is implicitly specified using the *transfer_image* statement, as the relevant transformations are automatically effected when one image is transfered to an image which is of a different type (or size).
- The third *transfer_image* statement listed in procedure *gen_and_sel_contours*, from a grey-level image to zero-crossing array image, also generates the zero-crossing contours together with the associated orientation and slope contours.
- The use of procedures makes the code more flexible. The *gen_and_sel_contours* routine may be used to produce different multiple spatial channels.

The VISICL program is as follows:

```
/* This VISICL command file generates intensity discontinuites from a stored intensity image. */

VAR gaussian_sigma, min_slope; /* Integer variables. */

PROCEDURE set_up_status();

/* Add images of the required types in the various pyramids. */

    add_image (1, "External", Framestore, 2, "Display", 0,0,511,511,pcplus);
    add_image (1, "External", Framestore, 4, "File", 0,0,255,255,VFS, "stored.vfs");
    add_image (2, "Grey Level", Intensity, 4,"Array", 0,0,255,255);
    add_image (4, "Intensity Discontinuities", Zero_crossing, 4, "Array", 0,0,255,255);
    add_image (5, "Intensity Discontinuities", Contour, 4, "Contours", 0,0,255,255);
    add_image (6, "Intensity Discontinuities", Slope, 4,"Contours", 0,0,255,255);
    add_image (7, "Intensity Discontinuities", Orientation, 4, "Contours", 0,0,255,255);

ENDPROC;

PROCEDURE gen_and_sel_contours(index, sigma, slope);

/* Procedure to generate and select intensity discontinuity contours. */

    transfer_image(1,4, index+2,4); /* Original image to grey-level intensity image. */
    transfer_image(index+2,4, 1,2); /* Grey-level to display. */

    /* Grey-level to zero-crossing array and contours: */

    transfer_image(index+2,4, index+4, 4, SIGMA = sigma, CONTOUR, index+5,4,
                                                    index+6,4, index+7,4);
```



```
    transfer_image(index+4,4, 1,2);      /* Result to display. */

    /* Select contours over a minimum slope threshold. */

    deselect_all_contours(index+5,4);
    or_select_contours(index+5,4, MEAN_SLOPE = slope , 500);
    /* Display the remaining contours. */

    transfer_image(index+5,4, index+4,4, GREY_LEVEL = 255);
    transfer_image(index+4,4, 1,2); /* Zero-crossing to display. */

ENDPROC;

/* MAINLINE. */

min_slope := 6;
gaussian_sigma := 3;

set_up_status();

specify_window(1,2, 0,0,255,255);
gen_and_sel_contours(0, gaussian_sigma min_slope);
```

This example is representative only of the style of a VISICL program. The repertoire of functional primitives is quite large. For specific details, refer to the Backus-Naur Form (BNF) syntax of VISICL at the end of the chapter.

10.4 Control of peripheral devices.

VISICL may also be employed to control peripheral devices. Examples include a framestore (for image acquisition and display), a robot arm, an AGV, and a X-Y table. One use of framestore control (i.e. image display) was demonstrated in the previous example through the use of *transfer_image*, but additional facilities are available which are shown here (e.g. the functions which activate continuous capture and image acquisition). Control of the AGV and the X-Y table are performed using very simple commands which are documented in the syntax of VISICL, at the end of this chapter.

VISICL also provides a programming interface for a UMI RTX six-degree-of-freedom robot. Programming is effected in a Cartesian frame of reference (as opposed to a joint-space frame of reference) whereby the robot program identifies the position and orientation (pose) of the wrist of the robot. The wrist pose is specified using *frames* represented by *homogeneous transformations*, i.e., by specifying the transformation necessary to translate an X-Y-Z coordinate frame from a position coincident with the 'real-world' reference frame to that position and orientation (pose) which one wishes for the wrist. Obviously, an X-Y-Z coordinated reference frame is

assumed to be affixed to the robot wrist.

The example which follows moves the robot through a sequence of nine positions, in order to allow the subsequent computation of depth from camera motion. The primitive *move_robot* was specifically developed for this task, but it should be noted that general robot control (i.e. in Cartesian space) is supported (through the use of homogeneous transformations) by several functional primitives (for example the *move* primitive which is passed a homogeneous transform and it then drives the robot end-effector to the position specified by that frame, if it specifies a wrist coordinate frame within the work envelope. For further details on this approach to robot programming see [1].

The example which follows is drawn from the program which computes optical flow from camera motion (see chapter 5).

```
first := 1;
last := 9;
framestore := 1; /* Framestore pyramid number. */
fs_level := 2;   /* Framestore image level. */
intensity := 20; /* Base index for intensity pyramid.*/
fp_dist := 500;  /* The radial fixation distance for the camera motion. */
theta := 2;      /* The angular step between frames in the motion sequence. */
```

•
•
•

```
set_up_images(); /* Generate the various required images. */

init_robot(); /* Initialise robot arm. */

/* Window the center of the framestore image only: */
specify_window(framestore,fs_level,128,128,383,383);

/* Acquire images and transfer to intensity */

/* Move to first position and wait for a user response. */
continuous_capture(framestore,fs_level);
move_robot(first, theta, fp_dist, 9); /* Move to the first position in the motion sequence. */
pause(); /* Wait for a user response. */

/* Obtain first image. */
frame_grab(framestore,fs_level); /* Snap image. */
transfer_image(framestore,fs_level, intensity+first,4);
transfer_image(intensity+first,4,vfs+first,4); /* Save image. */
```



```
/* Obtain the rest of the sequence of images. */
FOR i := first+1 TO last
DO
    continuous_capture(framestore,fs_level);
    move_robot(i, theta, fp_dist, 9); /* Move to position. */
    frame_grab(framestore,fs_level); /* Snap image. */
    transfer_image(framestore,fs_level, intensity+i,4);
    transfer_image(intensity+i,4,vfs+i,4); /* Save image. */

ENDFOR;

reset_robot(); /* Stow the robot arm in the safe position. */
```

•
•
•

10.5 More Advanced Image Processing and Analysis.

VISICL provides many additional facilities; in particular, it supports all of the processing and analysis operations described in this book. To give an idea of how some of these operations were implemented, five examples are given in this section which cover the following topics:

1. The computation of stereo disparity.
2. The computation of depth based on optical flow.
3. Generation of the raw primal sketch.
4. Camera calibration.
5. Object recognition.

10.5.1 Stereo computation.

The stereo disparity primitive computes disparity on the basis of a number of different spatial channels (i.e. convolutions and zero-crossings which are computed using different effective sigmas) derived from two stereo images (see chapter 4). As it is based on the 'coarse-to-fine' hierarchical strategy involving stereo pairs of convolution and zero-crossing images at three resolutions, the primitive requires a very large number of parameters. Apart from the images, these include the camera parameters, the standard deviation (sigma) of the Laplacian of Gaussian filter, a flag indicating that the coarse-to-fine strategy is to be followed, and the level at which it is to begin:

```
•
•
•

stereo_disparity(sig*40,no_of_steps,pyramidal_flag,threshold,
                 average_disparity,focal,camera_dist,
                 angle_in_mil_radians,
                 contour+1,first_level,contour+2,first_level,
                 convolution+1,first_level,convolution+2,first_level,
                 contour+3,second_level,contour+4,second_level,
                 convolution+3,second_level,convolution+4,second_level,
                 contour+5,third_level,contour+6,third_level,
                 convolution+5,third_level,convolution+6,third_level,
                 intensity+5,third_level,intensity+6,third_level,
                 zc+5,third_level,zc+6,third_level,
                 intensity+7,third_level,disparity,third_level);
```

10.5.2 Optical flow computation.

In some cases the combination of functional primitives is quite complex as can be seen in the central part of a VISICL program for computing optical flow from camera motion (see chapter 5):

```
•
•
•

PROCEDURE estimate_depth(id_convol, id_contour, id_zc, id_intensity,
                         id_depth, id_range, id_display, id_level,
                         num_frames, theta_rot, fp_distance,
                         id_sigma, id_focal);

VAR vel_pyr, window, scale;

vel_pyr := 800;
window := 5;
scale := 1;

set_up_velocity(vel_pyr, id_level);

/* evaluate the orthogonal component of velocity for all images */

FOR i := first+2 TO last-2
DO
    orthogonal_component(id_contour+i,id_level, vel_pyr+i,id_level, id_convol+i-2,id_level,
```



```
                         id_convol+i-1,id_level, id_convol+i+1,id_level, id_convol+i+2,id_level,
                         SIGMA = id_sigma);

    link_trajectory(vel_pyr+i,id_level, id_display,id_level,
                    1, 6, 255,CLEAR_DESTINATION);
    j := i-1;

    x1 := (j/3) * 170;
    y1 := (j - ((j/3) * 3)) * 170;
    x2 := x1 + 169;
    y2 := y1 + 169;
    specify_window(framestore,fs_level, x1, y1, x2, y2);

    transfer_image(id_display,id_level, id_display,6);
    transfer_image(id_display,6, framestore,fs_level);

    /* evaluate the actual velocity for all possible images */
    /* and also perform contour matching */

    general_camera_motion(vel_pyr+i,id_level, 0, theta_rot, 0,
                          fp_distance,fp_distance, id_focal,
                          id_contour+i+1,id_level, id_sigma+2);

    link_trajectory(vel_pyr+i,id_level, id_display,id_level, 1, 6, 255,CLEAR_DESTINATION);
    j := i-1;

    x1 := (j/3) * 170;
    y1 := (j - ((j/3) * 3)) * 170;
    x2 := x1 + 169;
    y2 := y1 + 169;
    specify_window(framestore,fs_level, x1, y1, x2, y2);

    transfer_image(id_display,id_level, id_display,6);
    transfer_image(id_display,6, framestore,fs_level);

ENDFOR;

link_trajectory(vel_pyr+3,id_level, id_display,id_level, 1, 6, 255,CLEAR_DESTINATION);

specify_window(framestore,fs_level, 0,0,255,255);
transfer_image(id_convol+3,id_level, framestore,fs_level);
specify_window(framestore,fs_level, 0,256,255,511);
transfer_image(id_zc+3,id_level, framestore,fs_level);

/* now compute the depth when all the images are tracked */
theta_rot := 8; /* This angle was determined experimentally. */
compute_depth(id_contour+3,id_level, id_depth,id_level, 0,theta_rot,0,fp_distance,
              fp_distance, id_focal, num_frames, scale);
```

```
link_trajectory(vel_pyr+3,id_level, id_display,id_level,
                num_frames, 6, 255,CLEAR_DESTINATION);

specify_window(framestore,fs_level, 256,0,511,255);
transfer_image(id_display,id_level, framestore,fs_level);
specify_window(framestore,fs_level, 256,256,511,511);
transfer_image(id_depth,id_level, id_zc,id_level, GREY_LEVEL = 0);
transfer_image(id_zc,id_level, framestore,fs_level);

contour_smoothing(id_depth,id_level, window);

specify_window(framestore,fs_level, 256,0,511,255);
ENDPROC;
```

•
•
•

```
estimate_depth(convolution, contour, zc, intensity,
               depth, range, display, level,
               frames, theta, fp_dist,
               sigma, focal);
```

10.5.3 Raw primal sketch generation.

The raw primal sketch, as documented in Chapter 6, is not generated through the use of *transfer_image*. Rather it is supported explicitly through several special purpose primitives:

•
•
•

```
specify_window(1,2, 0, 0, 255, 255); /* Specify display window. */
transfer_image(1,4,1,2); /* Display original image. */

/* First spatial channel. */
specify_window(1,2, 0, 256, 255, 511); /* Specify display window. */
sigma := 3;
gen_and_sel_contours(0,sigma,min_slope);

/* Second spatial channel. */
specify_window(1,2, 256, 0, 511, 255); /* Specify display window. */
```



```
sigma := 6;
gen_and_sel_contours(10,sigma2,min_slope);

specify_window(1,2, 256, 256, 511, 511); /* Specify display window. */
coincident_contours(17,4,7,4); /* Compute spatially coincident intensity discontinuity contours. */
transfer_image(5,4, 4,4, GREY_LEVEL = 255);
transfer_image(4,4, 1,2); /* Display result. */

min_edge_length := 8;
orientation_leeway := 30;
raw_primal_sketch(5,4, min_edge_length, orientation_leeway, 100,0,0,10,0,1);
draw_RPS_primitives(3,4);
specify_window(1,2, 256, 0, 511, 255); /* Specify display window. */
transfer_image(3,4, 1,2); /* Display result. */
```

10.5.4 Camera calibration.

An essential component of any passive vision system which computes the 3-D structure of a scene is calibration of the camera(s). Camera calibration is performed, in *VIS à VIS*, using a single image of a single two-plane calibration object. For example, the listing which follows is drawn from a program which simply calibrates a camera and saves the resultant model.

```
CAMERA~CAMERA; /* a camera model data-type */
```

•
•
•

```
continuous_capture(1,2);
pause; /* Wait for user to place the calibration grid in place. */

frame_grab(1,2); /* Snap calibration image. */
transfer_image(1,2, 3,2); /* Framestore to intensity image. */

calibrate_camera(~CAMERA,60,60, 3,2, 1,2, 5,2, 4,2);

SAVE_CAMERA(~CAMERA,"calib.cam");
```

10.5.5 Object Recognition.

Object recognition, in *VIS à VIS*, is addressed through the comparison 3-D surface-based models (see chapter 9), and is equated to the task of comparing a single

viewed model (i.e. the 3-D structure visible from a given viewpoint) with a number of theoretical 3-D models which are known *a priori*. The *viewed* model may be generated from an arbitrary view of a theoretical model, from actively sensed range data or from passively generated range data (see chapters 5 and 8). The *known* models are specified using a simple CAD specification technique (again, see chapter 8, and see the *BUILD_MODEL* statements in the listing which follows). The example which follows is part of a VISICL program for recognising objects from range maps which were interpolated from range data computed using optical flow derived from camera motion. Some of the results of this program are shown at the end of chapter 9.

```
/* variables of type CAMERA, MODEL, and INTEGER, respectively */

CAMERA  ~CAMERA, ~CAMERA2;
MODEL   ¬MODEL, ¬GAUSSIAN, ¬MODELS, ¬RESULT, ¬DUMMY;
VAR     TESS, DIST1, DIST2, DIST3, RAD1, RAD2, RAD3, RAD4, display_level;

    •
    •
    •

/* Build a Gaussian Sphere - so that EGIs may be considered. */

BUILD_MODEL(¬GAUSSIAN , ¬DUMMY , "GAUSSIAN." ,
            1 , 4 , 70 , 2 ,
            0 );

/* Load the camera model with which to view the models. */

LOAD_CAMERA(~CAMERA,"range.cam");

/* Build the known/CAD models. */

TESS := 150; /* No. of tesselations. */
BUILD_MODEL(¬MODELS , ¬GAUSSIAN , "CONE." ,
            4 , 1 , -44 , 102 , TESS , 2 , 44 ,
              1 , -44 , 98 , TESS , 2 , 40 ,
            3 , 1 , 2 , 0 , 1 , 3 , 0 , 3 , 4 , 1 );

    •
    •
    •
```



```
BUILD_MODEL(¬MODELS , ¬GAUSSIAN , "BOOK." ,
            2 , 3 , -15 , 120 , 180 , 3 , 15 , 120 , 180 ,
            3 , 1 , 2 , 0 , 1 , 1 , 0 , 2 , 2 , 0 );

/* Load the view to be recognised. */
/* In this instance that view is a passively generated range model. */

EXTRACT_RANGE_MODEL(¬MODEL , ~CAMERA , "passive.txt" ,
                    2 , 8 , 1 , 1 , 100 );

/* Recognise the selected instance through comparison with*/
/* the known (CAD) models.                                */

RECOGNISE_VIEW(¬MODEL, ~CAMERA, ¬GAUSSIAN, ¬MODELS, ¬RESULT, ~CAMERA2,
               2,4,7,4,5,4, 3,4,8,4,6,4, 9,4,10,4,4,4,
               display_level, 1,2);
```

10.6 Achieving Parallelism with VISICL.

VIS à VIS is written in such a way as to be essentially hardware independent for vision tasks. Where hardware dependencies do occur in, for example, I/O modules, (low-level) hardware attributes are effectively hidden in driver modules with a well-defined application interface. However since conventional single CPU style computer architectures are I/O bandwidth-limited, as well as limited in overall CPU performance, parallelism was considered to be the most useful approach to exploit in the context of both I/O and CPU bound applications.

The implementation of *VIS à VIS* running on parallel transputer hardware supports a very coarse granularity parallelism closely following the model of parallel processing that the transputer supports in hardware. The transputer is a general purpose processor with a 4GB linear address space. It has a well defined model for both single processor and multi-processor parallelism.

A network of co-operating *VIS à VIS* processes can be built up through a remote procedure call mechanism. Image data is explicitly exchanged among the processes in a manner similar to the previously described image transfer mechanism.

These facilities are covered in detail in the next chapter.

10.7 Syntax for VISICL.

The syntax of VISICL is presented, in Backus Naur Format (BNF), in this section. It is provided so that the reader can get a more rigorous definition of the language and also to give an impression of the scope of VISICL and, indeed, the functionality of the *VIS à VIS* system. Syntactic constructs are enclosed in angle brackets < >, terminal symbols are written without such brackets, alternatives are indicated using the | symbol, and repetition (zero or more times) is denoted using braces { }.

```
<visicl_program> ::= { <definition> ; } { <statement> }

<definition> ::= <variable_definition> |
                 <frame_definition> |
                 <camera_definition> |
                 <model_definition> |
                 <procedure_definition>

<variable_definition> ::= VAR <variable> {, <variable>}

<frame_definition> ::= FRAME <frame_variable> {, <frame_variable>}
<frame_variable> ::= ^<variable>

<camera_definition> ::= CAMERA <camera_variable> {, <camera_variable>}
<camera_variable> ::= ~<variable>

<model_definition> ::= MODEL <model_variable> {, <model_variable>}
<model_variable> ::= ¬<variable>

<procedure_definition> ::= PROCEDURE <procedure_name> ( { <parameter_list> } )
                           <procedure_body>
                           ENDPROC

<procedure_name> ::= <variable>

<parameter_list> ::= { <variable_list> }

<variable_list> ::= <variable> { , <variable> }

<procedure_body> ::= { <definition> ;}{ <statement> }

<procedure_call> ::= <procedure_name> ( { <parameter_or_value_list> } )

<parameter_or_value_list> ::= { <variable_or_value_list> }

<variable_or_value_list> ::= <variable_or_value> {, <variable_or_value> }

<variable_or_value> ::= <integer> |
                        <variable> |
                        <gripper_position_factor> |
                        <infer_focal_length_factor>
    <gripper_position_factor> ::= GRIPPER_POSITION
    <infer_focal_length_factor> ::= INFER_FOCAL_LENGTH( <camera_variable>, <si_or_sj> )
        <si_or_sj> ::= "si" | "sj"

<statement> ::= <repeat_statement> |
                <while_statement> |
                <assignment_statement> |
                <frame_assignment_statement> |
                <if_statement> |
                <for_statement> |
                <functional_primitive> |
                <procedure_call> ;
```



```
<repeat_statement> ::= REPEAT
                       { <statement> }
                       UNTIL <relational_expression>

<relational_expression> ::= <factor> <relational_operator> <factor>

<factor> ::= <integer> |
             <variable> |
             <gripper_position_factor> |
             <infer_focal_length_factor> |
             ( <expression> )

<integer> ::= { <digit> }

<variable> ::= <letter> { <letter> | <digit> }

<expression> ::= <term> |
                 -<term> |
                 <expression> + <term> |
                 <expression> - <term>

<term> ::= <factor> |
           <term> * <factor> |
           <term> / <factor>

<letter> ::= A | B | - - - | Y | Z | a | b | - - - | y | z

<digit> ::= 0 | 1 | - - - | 9

<relational_operator> ::= = | <> | <= | >= | < | >

<if_statement> ::= IF <relational_expression>
                   THEN { <statement> }
                   { ELSE { <statement> } }
                   ENDIF

<assignment_statement> ::= <variable> := <expression>

<for_statement> ::= FOR <variable> := <expression> TO <expression>
                    DO { <statement> }
                    ENDFOR

<while_statement> ::= WHILE <relational_expression>
                      DO { <statement> }
                      ENDWHILE

<case_label> ::= <variable> | <digit>

<frame_assignment_statement> ::= <frame_variable> := <frame_expression>

<frame_expression> ::= <frame_factor> { * <frame_factor> }

<frame_factor> ::= <frame_variable> |
                   <translation_factor> |
                   <rotx_factor> |
```

```
                    <roty_factor> |
                    <rotz_factor> |
                    <inv_factor> |
                    <rpy_factor> |
                    <rtx_curr_position_factor> |
                    <camera_frame_factor> |
                    <load_frame_factor> |
                    <motion_position_factor>
<translation_factor> ::= TRANS( <x>, <y>, <z>)
        <x> ::= <variable_or_value>)
        <y> ::= <variable_or_value>)
        <z> ::= <variable_or_value>)
<rotx_factor> ::= ROTX( <angle> )
<roty_factor> ::= ROTY( <angle> )
<rotz_factor> ::= ROTZ( <angle> )
        <angle> ::= <variable_or_value>
<inv_factor> ::= INV( <frame_expression> )
<rpy_factor> ::= RPY( <roll> , <pitch> , <yaw> )
<rtx_curr_position_factor> ::= RTX_CURR_POS
<camera_frame_factor> ::= CAMERA_FRAME( <camera_variable> )
<load_frame_factor> ::= LOAD_FRAME( <file_name> )
<motion_position_factor> ::= MOTION_POSITION( <focus_position>, <radius>,
                                <angle>, <no_positions>, <position>)
        <focus_position> ::= <frame_variable>
        <radius> ::= <variable_or_value>
        <angle> ::= <variable_or_value>
        <no_positions> ::= <variable_or_value>
        <position> ::= <variable_or_value>

<file_name> ::= <string_literal>

<string_literal> ::= " { <alphanumeric> } "

<alphanumeric> ::= <letter> | <digit> | .

<image> ::= <pyramid_number>,<image_level>
        <pyramid_number> ::= <expression>
        <image_level> ::= <expression>

<framestore_image> ::= <image>
<intensity_image> ::= <image>
<convolution_image> ::= <image>
<zero_crossing_image> ::= <image>
<contour_image> ::= <image>
<slope_image> ::= <image>
<orientation_image> ::= <image>
<disparity_image> ::= <image>
<velocity_image> ::= <image>
<depth_image> ::= <image>
<region_image> ::= <image>
<range_image> ::= <image>

<window_coordinates> ::= <top_l_h_coords>, <bottom_r_h_coords>
        <top_l_h_coords> ::= <expression>, <expression>
        <bottom_r_h_coords> ::= <expression>, <expression>
```



```
<functional_primitive> ::= <write_statement> |
                           <pause_statement> |
                           <terminate_visicl_statement> |
                           <echo_program_statement> |
                           <clear_int_image_statement> |
                           <fade_int_image_statement> |
                           <halt_statement> |
                           <stretch_range_statement> |
                           <delay_statement> |
                           <open_text_window> |
                           <prototype_statement> |
                           <add_image_statement> |
                           <delete_image_statement> |
                           <display_system_status_statement> |
                           <specify_window_statement> |
                           <transfer_image_statement> |
                           <par_transfer_image_statement> |
                           <init_framestore_statement> |
                           <framegrab_statement> |
                           <continuous_capture_statement> |
                           <select_video_statement> |
                           <select_all_contours_statement> |
                           <deselect_all_contours_statement> |
                           <or_select_contours_statement> |
                           <select_significant_contours_statement>|
                           <contour_interpolation_statement> |
                           <contour_smoothing_statement> |
                           <coincident_contours_statement> |
                           <raw_primal_sketch_statement> |
                           <choose_RPS_primitives_statement> |
                           <draw_RPS_primitives_statement> |
                           <select_RPS_primitives_statement> |
                           <list_RPS_information_statement> |
                           <load_rps_edges_statement> |
                           <edge_information_statement> |
                           <stereo_disparity_statement> |
                           <plot_optic_flow_vectors_statement> |
                           <orthogonal_component_statement> |
                           <egocentric_camera_motion_statement> |
                           <compute_depth_statement> |
                           <depth_uncertainty_statement> |
                           <link_trajectory_statement> |
                           <general_camera_motion_statement> |
                           <depth_from_motion_statement> |
                           <max_disparity> |
                           <load_camera_statement> |
                           <save_camera_statement> |
                           <theoretical_camera> |
                           <calibrate_camera_statement> |
                           <draw_camera_grid_statement> |
                           <define_camera_statement> |
                           <save_frame_statement> |
                           <initialise_statement> |
                           <move_statement> |
```

<home_statement> |
<go_to_initial_statement> |
<grasp_statement> |
<encoder_move_statement> |
<move_gripper_abs_statement> |
<move_wrist_abs_statement> |
<disable_statement> |
<enable_statement> |
<load_range_image_statement> |
<save_range_image_statement> |
<interpolate_range_data_statement> |
<range_modal_filter_statement> |
<range_median_filter_statement> |
<save_range_model_statement> |
<build_model_statement> |
<draw_model_statement> |
<shade_model_statement> |
<display_model_statement> |
<draw_depth_map_statement> |
<display_segmented_statement> |
<transform_model_statement> |
<user_select_model_statement> |
<extract_range_model_statement> |
<load_model_statement> |
<save_model_statement> |
<assign_gaussian_areas_statement> |
<assign_gaussian_visible_statement> |
<draw_gaussian_statement> |
<recognise_view_statement> |
<read_serial_statement> |
<agv_start_statement> |
<agv_drive_statement> |
<agv_init_statement> |
<x_y_table_home_statement> |
<x_y_table_move_statement> |
<put_iconic_image_statement> |
<get_iconic_image_statement> |
<build_cont_descriptors_statement> |
<sleep_statement> |
<remote_execute_statement> |
<put_image_statement> |
<get_image_statement> |
<put_var_statement> |
<get_var_statement> |
<rvp_time_statement> |
<par_statement>

<write_statement> ::= WRITE ({ <variable> |
<string_literal> })

<pause_statement> ::= PAUSE

<terminate_visicl_statement> ::= TERMINATE_VISICL

<echo_program_statement> ::= ECHO_PROGRAM (<flag>)
<flag> ::= <expression>

<clear_int_image_statement> ::= CLEAR_INT_IMAGE(<intensity_image>
<fade_int_image_statement> ::= FADE_INT_IMAGE(<intensity_image>, <divisor>, <knock_off>)
<divisor> ::= <expression>
<knock_off> ::= <expression>

<halt_statement> ::= HALT

<stretch_range_statement> ::= STRETCH_RANGE(<range_image>, <intensity_image>)

<delay_statement> ::= DELAY(<period>)
<period> ::= <expression>

<open_text_window> ::= OPEN_TEXT_WINDOW (<window_name>, <window_length>, <window_depth>,
<window_x_pos>, <window_y_pos>)
<window_name> ::= <string_literal>
<window_length> ::= <expression>
<window_depth> ::= <expression>
<window_x_pos> ::= <expression>
<window_y_pos> ::= <expression>

<prototype_statement> ::= PROTOTYPE_ALGORITHM (<prototype_number>)
<prototype_number> ::= <integer> between 0 and 18

<add_image_statement> ::= ADD_IMAGE(<pyramid_number>, <pyramid_description>,
<image_type>, <image_level>, <image_description>,
<window_coordinates> {, <framstore_type>
{, <vfs_filename>} })
<pyramid_description> ::= <string_literal>
<image_type> ::= FRAMESTORE | INTENSITY | CONVOLUTION | ZERO_CROSSING |
CONTOUR | SLOPE | ORIENTATION | DISPARITY |
VELOCITY | DEPTH | REGION | RANGE
<image_description> ::= <string_literal>
<framestore_type> ::= PCPLUS | EIDOBRAIN | MICROEYE | VICOM | FG100AT | VFS
<vfs_filename> ::= <string_literal>

<delete_image_statement> ::= DELETE_IMAGE (<image>)

<display_system_status_statement> ::= DISPLAY_SYSTEM_STATUS

<specify_window_statement> ::= SPECIFY_WINDOW(<image>, <window_coordinates>)

<transfer_image_statement> ::= TRANSFER_IMAGE(<source_image>, <destination_image>
{ , SIGMA = <sigma> } { , GREY_LEVEL = <grey_level> }
{ , CONTOUR , <contour_image> , <slope_image> ,
<orientation_image> {, <convolution_image> } }
{ , ICONIC , <zero_crossing_image> })
<source_image> ::= <pyramid_number>,<image_level>
<destination_image> ::= <pyramid_number>,<image_level>
<sigma> ::= <expression>
<grey_level> ::= <expression>

*image>
*e> , <bar_match_percentage>
*ce> , <max_blob_dimension>
}

*primitive_type>
<primitive_type> })
OB | BAR

*nsity_image>)

*old> , <min_line_length>
*dimension>)

*e>)

*umber_of_steps>, <pyramidal_flag>,

*gle_between_cameras>,
*r_image>,
*nvolution_image>,
*s_contour_image>,
*n_res_convolution_image>,
*our_image>,
*convolution_image>,
*nage>,
*age>,

*S (<velocity_image>

<scale> ::= <expression>

<max_disparity> ::= MAX_DISPARITY(<disparity_image>, <probability_image>, <threshold>,
<result>)
<probability_image> ::= <intensity_image>
<threshold> ::= <expression>
<result> ::= <variable>

<load_camera_statement> ::= LOAD_CAMERA(<camera_variable>, <file_name>)

<save_camera_statement> ::= SAVE_CAMERA(<camera_variable>, <file_name>)

<theoretical_camera> ::= THEORETICAL_CAMERA(<camera_variable> , <frame_expression>,
<f>, <si>, <sj>, <ic>, <jc>)
<f> ::= <expression>
<si> ::= <expression>
<sj> ::= <expression>
<ic> ::= <expression>
<jc> ::= <expression>

<calibrate_camera_statement> ::= CALIBRATE_CAMERA(<camera_variable>,
<grid_length>, <grid_width>, <calibration_image>,
<framestore_image>, <hough_table>, <display_image>)
<grid_length> ::= <variable_or_value>
<grid_width> ::= <variable_or_value>
<calibration_image> ::= <intensity_image>
<display_image> ::= <intensity_image>
<hough_table> ::= <range_image>

<draw_camera_grid_statement> ::= DRAW_CAMERA_GRID(<camera_variable>,
<grid_length>, <grid_width>, <intensity_image>)
<grid_length> ::= <variable_or_value>
<grid_width> ::= <variable_or_value>

<define_camera_statement> ::= DEFINE_CAMERA(<camera_variable>, <frame_expression>)

<save_frame_statement> ::= SAVE_FRAME(<frame_expression> , <file_name>)

<initialise_statement> ::= INITIALISE

<move_statement> ::= MOVE(<frame_variable>, <gripper_position>)
<gripper_position> ::= <variable_or_value>

<home_statement> ::= HOME

<go_to_initial_statement> ::= GO_TO_INITIAL

<grasp_statement> ::= GRASP(<gripper_distance>)
<gripper_distance> ::= <expression>

<encoder_move_statement> ::= ENCODER_MOVE(<zed>, <shoulder>, <elbow>,
<yaw>, <pitch>, <roll>, <gripper>)
<zed> ::= <expression>
<shoulder> ::= <expression>
<elbow> ::= <expression>

<yaw> ::= <expression>
<pitch> ::= <expression>
<roll> ::= <expression>
<gripper> ::= <expression>

<move_gripper_abs_statement> ::= MOVE_GRIPPER_ABS(<x> , <y> , <z>
<roll> , <pitch> , <yaw> , <gripper_position>)
<move_wrist_abs_statement> | ::= MOVE_WRIST_ABS(<x> , <y> , <z> , <roll>
, <pitch> , <yaw> , <gripper_position>)
<x> ::= <expression>
<y> ::= <expression>
<z> ::= <expression>
<roll> ::= <expression>
<pitch> ::= <expression>
<yaw> ::= <expression>
<gripper_position> ::= <expression>

<disable_statement> ::= DISABLE

<enable_statement> ::= ENABLE

<load_range_image_statement> ::= LOAD_RANGE_IMAGE(<range_image>, <file_name>)

<save_range_image_statement> ::= SAVE_RANGE_IMAGE(<range_image>, <file_name>)

<interpolate_range_data_statement> ::= INTERPOLATE_RANGE_DATA(<depth_image>,
<resultant_range_image>, <range_image>,
<intensity_image>, <framestore_image>,
<maximum_distance>, <sine_iterations>)
<resultant_range_image> ::= <range_image>
<maximum_distance> ::= <variable_or_value>
<sine_iterations> ::= <variable_or_value>

<range_modal_filter_statement> ::= RANGE_MODAL_FILTER(<range_image>,
<resultant_range_image>, <intensity_image>,
<framestore_image>, <filter_width>,
<smoothing_width>, <camera>)
<resultant_range_image> ::= <range_image>
<filter_width> ::= <variable_or_value>
<smoothing_width> ::= <variable_or_value>

<range_median_filter_statement> ::= RANGE_MEDIAN_FILTER(<range_image>,
<resultant_range_image>, <intensity_image>,
<framestore_image>, <filter_width>)
<resultant_range_image> ::= <range_image>
<filter_width> ::= <variable_or_value>

<save_range_model_statement> ::= SAVE_RANGE_MODEL(<range_image>,
<camera_variable>, <file_name>, <sub_sampling>)
<sub_sampling> ::= <variable_or_value>

<build_model_statement> ::= BUILD_MODEL(<model_list>, <gaussian>, <model_name>,
<no_components> {, <component_description>},
<no_connections> {, <connection_description>}
{, <z_angle>, <x_angle>, <x_translation>,
<no_components> {, <component_description>},

<no_connections> {, <connection_description>} })
<model_list> ::= <model_variable>
<gaussian> ::= <model_variable>
<model_name> ::= <string_literal>
<no_components> ::= <variable_or_value>
<component_description> ::= <ring_component> |
<point_component> |
<box_component> |
<sphere_component>
<ring_component> ::= 1, <axis_position>, <radius>, <no_tessellations>
<point_component> ::= 2, <axis_position>
<box_component> ::= 3, <axis_position>, <height>, <width>
<sphere_component> ::= 4, <radius>, <level_of_detail>
<axis_position> ::= <variable_or_value>
<radius> ::= <variable_or_value>
<no_tessellations> ::= <variable_or_value>
<height> ::= <variable_or_value>
<width> ::= <variable_or_value>
<level_of_detail> ::= <variable_or_value>
<no_connections> ::= <variable_or_value>
<connection_description> ::= <surface_no>, <surface_no>, <towards_origin>
<surface_no> ::= <variable_or_value>
<towards_origin> ::= <variable_or_value>
<z_angle> ::= <variable_or_value>
<x_angle> ::= <variable_or_value>
<x_translation> ::= <variable_or_value>

<draw_model_statement> ::= DRAW_MODEL(<model_variable>, <camera_variable>,
<intensity_image>, <range_image>)

<shade_model_statement> ::= SHADE_MODEL(<model_variable>, <camera_variable>,
<intensity_image>, <range_image>)

<display_model_statement> ::= DISPLAY_MODEL(<model_variable>, <camera_variable>,
<intensity_image>, <range_image>,
<framestore_image>)

<draw_depth_map_statement> ::= DRAW_DEPTH_MAP(<model_variable>, <camera_variable>,
<intensity_image>, <range_image>)

<display_segmented_statement> ::= DISPLAY_SEGMENTED(<model_variable>, <camera_variable>,
<intensity_image>, <range_image>,
<framestore_image>)

<transform_model_statement> ::= TRANSFORM_MODEL(<model_variable>, <transform>)
<transform> ::= <frame_variable>

<user_select_model_statement> ::= USER_SELECT_MODEL(<models>, <camera_variable>,
<intensity_image>, <range_image>,
<framestore_image>, <result>)
<models> ::= <model_variable>
<result> ::= <model_variable>

<extract_range_model_statement> ::= EXTRACT_RANGE_MODEL(<model_variable>,
<camera_variable>, <file_name>, <sub_sampling>,
<smoothing_iterations>, <multiple_objects>,

<center_model>, <scaling_factor>)
<sub_sampling> ::= <variable_or_value>
<smoothing_iterations> ::= <variable_or_value>
<multiple_objects> ::= <variable_or_value>
<center_model> ::= <variable_or_value>
<scaling_factor> ::= <variable_or_value>

<load_model_statement> ::= LOAD_MODEL(<model_variable>, <file_name>)

<save_model_statement> ::= SAVE_MODEL(<model_variable>, <file_name>)

<assign_gaussian_areas_statement> ::= ASSIGN_GAUSSIAN_AREAS(<model_variable>,
<gaussian>)
<gaussian> ::= <model_variable>

<assign_gaussian_visible_statement> ::= ASSIGN_GAUSSIAN_VISIBLE(<model_variable>,
<gaussian>)
<gaussian> ::= <model_variable>

<draw_gaussian_statement> ::= DRAW_GAUSSIAN(<gaussian>, <camera_variable>,
<intensity_image>)
<gaussian> ::= <model_variable>

<recognise_view_statement> ::= RECOGNISE_VIEW(<viewed_model>, <camera>
<gaussian>, <known_models>, <result>,
<result_camera>, <viewed_tilt_image>,
<viewed_orientation_image>, <viewed_range_image>,
<known_orientation_image>, <known_orientation_image>,
<known_range_image>, <sub_sampled_tilt_image>,
<sub_sampled_orientation_image>, <scrap_image>,
<display_level>, <framestore_image>)
<viewed_tilt_image> ::= <intensity_image>
<viewed_orientation_image> ::= <intensity_image>
<viewed_range_image> ::= <range_image>
<known_orientation_image> ::= <intensity_image>
<known_orientation_image> ::= <intensity_image>
<known_depth_image> ::= <range_image>
<sub_sampled_tilt_image> ::= <intensity_image>
<sub_sampled_orientation_image> ::= <intensity_image>
<scrap_image> ::= <intensity_image>
<display_level> ::= <variable_or_value>

<read_serial_statement> ::= READ_SERIAL(<result>)
<result> ::= <variable>

<agv_start_statement> ::= AGV_START

<agv_drive_statement> ::= AGV_DRIVE (<speed> , <direction>)
<video_input> ::= <speed>
<video_input> ::= <direction>

<agv_init_statement> ::= AGV_INITIALISE (<a>)

<x_y_table_home_statement> ::= X_Y_TABLE_HOME

```
<x_y_table_move_statement> ::= X_Y_TABLE_MOVE( <delta_x>, <delta_y> )
      <delta_x> ::= <expression>
      <delta_y> ::= <expression>

<put_iconic_image_statement> ::= PUT_ICONIC_IMAGE( <channel>, <image>, <command>,
                                 <parameter> )
      <channel> ::= <expression>
      <command> ::= <expression>
      <parameter> ::= <expression>

<get_iconic_image_statement> ::= GET_ICONIC_IMAGE( <channel>, <image>, <command>,
                                 <parameter> )
      <channel> ::= <expression>
      <command> ::= <expression>
      <parameter> ::= <expression>

<build_cont_descriptors_statement> ::= BUILD_CONT_DESCRIPTORS( <contour_image>,
                                       <other_chain_image> { , <other_chain_image> } )
      <other_chain_image> ::= <slope_image> | <orientation_image> | <velocity_image> |
                              <depth_image> | <disparity_image>

<sleep_statement> ::= SLEEP( <duration> )
      <duration> ::= <expression>

<remote_execute_statement> ::= REMOTE_EXECUTE ( <link_specification>, <procedure_call> )
      <channel> ::= <variable> | <expression>

<put_image_statement> ::= PUT_IMAGE( <channel>, <image>, <command>, <parameter> )
      <channel> ::= <expression>
      <command> ::= <expression>
      <parameter> ::= <expression>

<get_image_statement> ::= GET_IMAGE( <channel>, <image>, <command>, <parameter> )
      <channel> ::= <expression>
      <command> ::= <expression>
      <parameter> ::= <expression>

<put_var_statement> ::= PUT_VAR( <channel>, <command>, <parameter> )
      <channel> ::= <expression>
      <command> ::= <variable>
      <parameter> ::= <variable>

<get_var_statement> ::= GET_VAR( <channel>, <command>, <parameter> )
      <channel> ::= <expression>
      <command> ::= <variable>
      <parameter> ::= <variable>

<rvp_time_statement> ::= RVP_TIME( <comment> )
      <comment> ::= <string_literal>

<par_statement> ::= PAR(<link_specification>, <procedure_call>
                        { , <link_specification>, <procedure_call> } )
      <link_specification> ::= LINK0 | LINK1 | LINK2 | LINK3
```


The protocol for transfer of low-level information is specified by the following syntax:

```
<information_stream> ::= { <header> <cue_information> }
    <header> ::= <information_type> <channel_size> <resolution>
        <information_type> ::= EDGE
        <channel_size> ::= <real>
        <resolution> ::= <integer>
    <cue_information> ::= <line_information>
        <line_information> ::= <line> {<line>}
            <line> ::= <line_header> {<edge>}
                <line_header> ::= <initium> <terminus>
                    <max_strength>
                    <min_strength>
                    <mean_direction>
                    <standard_deviation_direction>
                    <number_of_edges>
                    <initium> ::= <x> <y> <z>
                        <x> ::= <integer>
                        <y> ::= <integer>
                        <z> ::= <integer>
                    <terminus> ::= <x> <y> <z>
                    <max_strength> ::= <integer>
                    <min_strength> ::= <integer>
                    <mean_direction> ::= <real>
                    <standard_deviation_direction> ::= <real>
                    <number_of_edges> ::= <integer>
                <edge> ::= <initium> <termination> <strength>
```

Bibliography

[1] Vernon, D. 1991, *Machine Vision,* Prentice-Hall International, London.

Chapter 11

Distributed Visual Processing; From *VIS* to *VIS*

Paul Healy

Four other Oysters followed them,
And yet another four;
And thick and fast they came at last,
And more, and more, and more ...

Lewis Carroll
Through the Looking Glass

11.1 Introduction

The spectrum of applications for which machine vision solutions can be provided at present is severely restricted by the limitations of hardware performance. Even where commercial solutions do exist a lack of flexibility and significant cost can limit the appeal to manufacturing industry.

This chapter describes the efforts which were made to make the original Virtual Image System (*VIS*) run on an expandable and reconfigurable parallel transputer-based platform. The vision system which was developed addresses the two key issues of performance and flexibility and we have dubbed it *VIS* $\vec{a}$ *VIS*, to reflect the manner in which the parallelism is effected, i.e., by passing messages — VISICL programs and image data — between multiple instantiations of *VIS*. But more of this later. First, let us look at the manner and context in which *VIS* $\vec{a}$ *VIS* was designed.