JARtool 2.0

JPL ADAPTIVE RECOGNITION TOOL

USER'S MANUAL

Developed by:

M.C. Burl, S. Mukhtar, J.C. Roden, M.P. Burl, P. Smyth, U.M. Fayyad

Machine Learning Systems Group
Information and Computing Technologies Research Section
Jet Propulsion Laboratory
California Institute of Technology

September 16, 1997

U.S. Government Sponsorship Acknowledged NAS7-1260

JPL - NASA

This document describes a system that was developed at the Jet Propulsion Laboratory/California Institute of Technology to provide scientists with tools to analyze large image databases. No representations are made about the suitability of this software for any purpose. It is distributed "as is" without express or implied warranty and without technical support. For further information, contact:

Dr. Michael C. Burl (818) 306-6422 burl@aig.jpl.nasa.gov

U.S. Government Sponsorship Acknowledged NAS7-1260

Typeset with LaTeX.

Contents

Part I

Reference Manual

Chapter 1

Introduction

JARtool 2.0 was developed at the Jet Propulsion Laboratory (JPL) as a prototype system for finding objects of interest in image databases. Improvements in acquisition and storage technology have resulted in image databases that are so large it is no longer feasible to manually examine each collected image. Scientists need automated tools if even a fraction of the data is to be analyzed.

Consider the Magellan mission to Venus. The objective of this mission was to provide global mapping of the planet's surface. The mission was highly successful achieving 98% coverage and returning *more data than all previous planetary missions combined*. Planetary geologists were understandably excited about the possibilities opened up by this new dataset, but they were also overwhelmed by the sheer volume of data.

One area of particular scientific interest on Venus is volcanism, which is the dominant geological process on the planet. Geologists predict based on previous low-resolution data that approximately one million small volcanoes are visible in the Magellan dataset. However, to manually catalog all of these would require an estimated ten to twenty man-years. The JARtool project was initiated to investigate automating the cataloging process through the use of pattern recognition and machine learning techniques.

The standard approach to a problem of this type would have been to develop hand-coded, volcano-specific detectors based on rules provided by the domain experts — the planetary geologists. However, there are several drawbacks to this approach: (1) geologists can readily identify examples of volcanoes, but they have difficulty verbalizing which characteristics of the image led to their decision, (2) even when expert rules are available (e.g., volcanoes have a circular planimetric outline), they are not easily translated into pixel-level constraints, and (3) the prob-

lem of visual detection of localized objects occurs in many domains including military surveillance, medical image anlysis, etc. With hand-coded features, a new system would need to be developed for each new problem domain.

Given these limitations, we elected to pursue a *learning from examples* approach in which the geologists' domain knowledge was captured, not through explicit rules, but through examples of the objects of interest. This, in fact, is the fundamental idea behind JARtool. A scientist (or other user) provides a set of training examples by using the graphical user interface (GUI) to label target objects within a set of images. During a learning phase, the system uses these examples to build an appearance model of the object. During a production phase, the learned model is applied to other images to find novel instances of the object.

Although JARtool was developed in the context of locating small volcanoes in the Magellan dataset, the fact that it is trainable, in principle, means that the system can be applied to other datasets merely by supplying the appropriate training examples. However, in practice this is not quite the case. The system uses as examples fixed-size regions centered on the object. Thus, if an object class exhibits large variations in scale, the JARtool approach will not work well. For the volcano problem we were fortunate: the volcanoes *do* vary significantly in scale, *but* a fixed-size area near the center is distinctive enough to support reliable detection. Analagous to the "limited variation in scale" assumption, the system works best if there is also limited variation in object orientation and illumination direction, but may work even if these conditions are not satisfied.

A recent improvement to the system was the addition of the *Postgres* database, which is used to maintain labeling results and meta-data about image sets. Scientists can interact with the database using both "pre-canned" and ad hoc SQL queries. The database adds a new dimension to the system as scientists can now formulate complex queries, such as "show me the number of volcanoes detected by the system in this area of the planet". Further, the database query tool contains specialized visualization features that connect the results of a query back to the JARtool display. Incidentally, JARtool is built on top of a public domain image display package and graphical user interface developed by the Smithsonian Astronomical Observatory (SAOtng, v1.7).

The manual is organized into three main parts: (I) Reference Manual (the part you are reading now), (II) Sample Session, and (III) Technical Appendices. Each of the parts contains one or more chapters. Chapter 2 covers the basics of the JARtool user interface including the physical layout of components and their functionality. Chapters 3– 10 cover each of the JARtool menu bar functions in more detail. Chapter 11 presents a detailed sample session that the user can follow to get a quick feel for the capabilities of JARtool. Finally, the technical appendices include chapters

on installing JARtool, the VIEW image format, the JTRI marker file format, learning parameters, production parameters, and a list of known bugs.

Much of the reference information contained in this manual is also available in hypertext form through the JARtool *Help* button. In addition, the hypertext documentation contains a list of frequently asked questions and answers (FAQ). The FAQ is not included in the hard-copy manual.

Chapter 2

JARtool Basics

To get started with JARtool, we recommend that you first study the information here and then jump ahead to the sample session described in Chapter 11. (Note: We assume you have already successfully installed JARtool on your system. If not, see the instructions in Appendix A.) After trying the examples in the sample session, return to Chapters 3– 10 for more detail on any of the system capabilities.

The JARtool interface consists of twelve components which are physically laid out as in Figure 2.1. Clockwise beginning from the left we have:

1. Accelerator Buttons
2. Menu Bar
3. Panner/Magnifier
4. Panner/Magnifier Popup Menu
5. Image Display Area
6. Coordinate Display
7. Coordinate Display Popup Menu
8. Colorbar
9. Colorbar Popup Menu
10. Text Information
11. Help Button
12. Marker Popup Menu

Note that the components designated as “popup menus” will not normally appear on the display. These are activated by using the mouse buttons as described later. Also, there are six other components that you will encounter while using JARtool.

- Parameter Input GUI's (PIG's)
- Entry Boxes
- Builder Buttons
- File Browser/File Selection Box (FSB)
- List Files
- Busy Boxes
- Marker Analysis Tasks

2.1 Accelerator Buttons

The buttons along the left side of the JARtool interface are called accelerator buttons. These buttons come in two varieties – true buttons and menu buttons. The menu buttons are distinguished by a series of dots following the button label. To use the true accelerator buttons, click with the left mouse button. To use the accelerator menu buttons, click *and drag* with the left mouse button. Release the button when the desired option is highlighted.

Load Image – This button provides a simple interface for loading images. Click and release the left mouse button to bring up a dialog box. You can then directly type the image filename into the pink entry box or click on the red builder button to the left of the entry box to invoke a file browser.

The preferred JARtool image format is VIEW format (described in Appendix B). Images in FITS (Flexible Image Transport System) format, the native format of SAOtng, can also be read and displayed in JARtool; however, the more interesting operations of learning and production, as well as many of the tools, will only work on VIEW format images.

The *Format Conversion* program under the *Tools* menu button can be used to convert a variety of image formats to VIEW format.

Save Image – This button provides a simple interface for saving images. Click and release the left mouse button to bring up a dialog box. You can then directly type the image filename into the pink entry box or click on the red builder button to the left of the entry box to invoke a file browser. Images can be saved in either VIEW or FITS format. Click with the left mouse button to select the desired format.

Print Image – This button provides an interface for printing images or saving as postscript. Type the print command into the dialog box. For example, to send

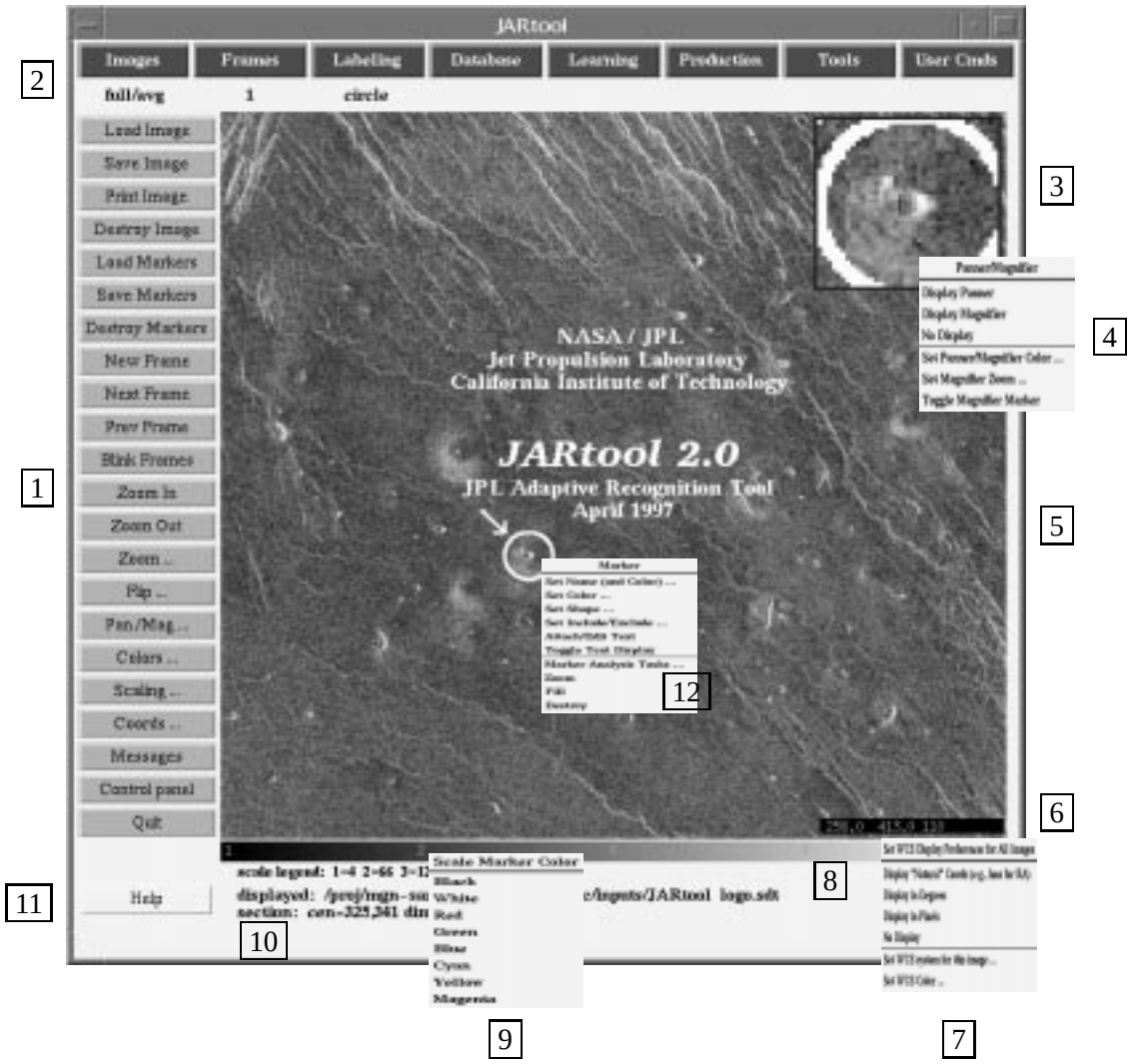


Figure 2.1: The JARtool graphical user interface.

the image to a printer called mammoth use lpr -Pmammoth. To save the image as a postscript file, just type the filename, e.g., tmp.ps

Destroy Image – This button destroys the currently displayed imaged. A popup dialog box will verify the request before destroying the image. There is a bug in the SAOtng system that causes the menu options under the menu bar buttons to disappear when an image is destroyed. The options can be restored by loading a new image into the display. (See Appendix F for a list of known

bugs.)

Load Markers – This button provides an interface for loading a JTRI format marker file onto the current display. Click and release the left mouse button to bring up a dialog box. You can then directly type the jtri filename into the pink entry box or click on the red builder button to the left of the entry box to invoke a file browser.

The new markers are merged with existing markers. If desired, use the *Destroy Markers* command before loading to clear pre-existing markers.

Save Markers – This button provides an interface for saving the currently displayed markers to a JTRI format file. Click and release the left mouse button to bring up a dialog box. You can then directly type the save filename into the pink entry box or click on the red builder button to the left of the entry box to invoke a file browser.

Destroy Markers – This button provides a shortcut for destroying all of the currently displayed markers. A popup dialog box will verify the request before destroying the markers.

To delete individual markers, move the cursor inside or over the marker and hit the delete or backspace key on the keyboard. See Chapter 5 for more information on markers.

New Frame – This button provides a shortcut for creating a new image frame. See the *Frames* menu bar button in Chapter 4 for more information.

Next Frame – This button provides a shortcut for displaying the next image frame in the JARtool image area. Invoking *Next Frame* from the highest numbered frame will go to the lowest numbered frame rather than create a new frame. (If you want to create a new frame, use the *New Frame* accelerator button.)

Prev Frame – This button provides a shortcut for displaying the previous image frame in the JARtool image area. Invoking *Prev Frame* from the lowest numbered frame will go to the highest numbered frame.

Blink Frames – This button provides a shortcut for blinking between all the displayed image frames. The control panel provides additional blink options including controls for the blink rate and the frames to blink between.

Zoom In – This button provides a shortcut for zooming in around the central point of the image display. *Zoom In* multiplies the current zoom setting by a factor of 2. See also the *Zoom*. . . accelerator menu button.

Zoom Out – This button provides a shortcut for zooming out around the central point of the image display. *Zoom Out* multiplies the current zoom factor by 1/2. See also the *Zoom*. . . accelerator menu button.

Zoom... – This button provides a menu of zooming options. Click and hold the left mouse button to see the options. The "Set Zoom Factor" option sets the zoom to a predetermined value (1, 2, or 4). A zoom factor of 1 is the normal display setting when an image is loaded. A zoom factor of 2 will blow up the central displayed area to fill the entire display. Use the panner to set the zoom center by adjusting the image so that the desired zoom center appears at the center of the display.

The "Current Zoom $\times$ Factor" option zooms in (factor > 1) or zooms out (factor < 1). The difference between this option and the "Set Zoom Factor" is that here the resulting zoom is relative to the current zoom setting rather than an absolute number.

Neither the "Set Zoom Factor" nor "Current Zoom x Factor" affect the function of the *Zoom In* or *Zoom Out* accelerator buttons, which are hard-wired to multiply the current zoom by 2 and 1/2, respectively.

Following a long sequence of pans and zooms, the option "Reset and Recenter Zoom" can be used to go back to the original view (like when the image was first loaded).

Flip... – This button provides a menu of image flipping options. Click and hold the left mouse button to see the options; they are all self-explanatory. Note that a flip operation can be undone by applying the same flip operation a second time.

Pan/Mag... – This button provides control over the panner/magnifier. The user can choose between panner, magnifier, and no display. The user can also control the magnification factor. More details on the panner/magnifier are given later in this chapter.

Colors... – This button provides a menu of colormap options. Click and hold the left mouse button to see the options; release on the desired option.

To change the display contrast and average brightness, simply click the right mouse button on the image display area and drag until you get the desired result. (You must click outside of any defined markers; otherwise, the program will think you are selecting a marker.)

Scaling... – This button provides a menu of scaling options. Click and hold the left mouse button to see the options; release on the desired option. These transform the pixel values before the colormap is applied. Normally, the default (linear scaling) should be used.

Coords... – This button provides a menu of coordinate display options. Click and hold the left mouse button to see the options; release on the desired option.

Normally, the default (pixels) should be used. There is also an option to turn off the display of the cursor coordinates.

Messages – This button brings up a text window which shows various messages generated by the JARtool and SAOtng procedures. If something doesn't work as expected, look here for additional information (and also in the window from which the JARtool program was invoked).

Control Panel – This button is a convenient SAOtng feature that allows the user to control many elements of the display from a single interface. It also includes more options to control blinking between frames.

Quit – This button exits from JARtool.

2.2 Menu Bar

The menu bar consists of the buttons near the top of the display. Clicking and holding the left mouse button on a menu bar item causes a pulldown menu to be displayed. To select an option from the pulldown menu, drag the mouse to the desired option and release. To avoid selecting any options, release the mouse button while the cursor is not on the menu.

The menu bar contains the following functions. These buttons will be discussed in greater detail in Chapters 3 – 10 respectively.

- Images
- Frames
- Labeling
- Database
- Learning
- Production
- Tools
- User Cmds

2.3 Panner/Magnifier

Near the upper right corner of the image area, there may be a small window with a special cursor. This window can serve as either a panner or magnifier. As a panner, the window shows a reduced size version of the entire image. The user can manipulate the view by clicking and dragging the cyan box in the panner window with the

left mouse button. (An alternative method of panning is to click the middle mouse button on the image display.)

As a magnifier, the window shows a closeup of the image area under the cursor. To switch between panner and magnifier, use the *Pan/Mag. . .* accelerator menu button or the *Panner/Magnifier Popup Menu*. (There are also options in these menus to turn off the panner/magnifier display and to set the magnification factor.)

2.4 Panner/Magnifier Popup Menu

In order to activate the panner magnifier popup menu, position the cursor inside the panner/magnifier and hold down the right mouse button. To select an option, drag the mouse so that the desired option is highlighted and then release the mouse button.

The available options enable a user to change the color of the panner/magnifier, toggle between panner and magnifier or completely hide the panner/magnifier window.

2.5 Image Display Area

When JARtool is started, the display is filled with an image of an area on Venus near latitude 30N, longitude 330E. The image also shows the JPL logo and the current JARtool release number. New images can be loaded from either the *Load Image* accelerator button or from the *Images* menu bar button. The size of the display can be increased by resizing the JARtool window.

2.6 Coordinate Display

Near the lower right corner of the image display area, there may be a small window in which cursor coordinates are displayed. This display can be turned off from the *Coords. . .* accelerator menu button or the*Coordinate Display Popup Menu*.

2.7 Coordinate Display Popup Menu

In order to activate this popup menu, position the cursor over the coordinate display. Drag with the right mouse button until the desired option is highlighted and then release the mouse button.

The available options enable the user to specify the units, the coordinate system to be used (if multiple coordinate systems are specified in the header of a FITS image), and the color of the coordinate display.

2.8 Colorbar

Slightly below the image display area there is an elongated bar that shows the mapping between pixel values and color. The image colormap can be modified by clicking and dragging the right mouse button within the image display area (outside of any marker symbols). To restore a standard colormap, use the *Colors. . .* accelerator menu button or the *Colorbar Popup Menu*.

2.9 Colorbar Popup Menu

In order to activate this popup menu, position the cursor above one of the numbers drawn on the colorbar. Hold down the right mouse button. To select a menu option, drag the mouse until the desired option in highlighted and then release the mouse button. The available options enable the user to specify the color of these numbers.

2.10 Text Information

Beneath the colorbar is a small area used for text display. Typically, this area contains the name of the displayed image and other information about the section of the image currently being displayed.

2.11 Help Button

At the lower left of the display is a help button that provides access to hypertext documentation. The documentation is divided into JARtool and SAOtng. Click with the left mouse button on any highlighted words to get more information.

2.12 Marker Popup Menu

In order to activate this popup menu, position the cursor inside a marker (now referred to as the active marker). Hold down the right mouse button. To select an option, drag the mouse until the desired option is highlighted and then release the mouse button. The available options enable the user to change the color, label, and

shape, as well as to associate text with a marker. The user can also perform some local operations on the part of the image which lies inside the marker. The available operations are present under the "Marker Analysis Tasks" menu, which is discussed later in this chapter.

2.13 Parameter Input GUI's (PIG's)

Virtually every JARtool function requires some parameters to be input by the user. These parameters are collected using a set of special purpose parameter input GUI's or *PIG's* for short. A typical PIG may contain entry boxes, radio buttons, checkbuttons, sliders, or other data entry widgets. Figure 2.2 shows the PIG used with the JARtool production process.

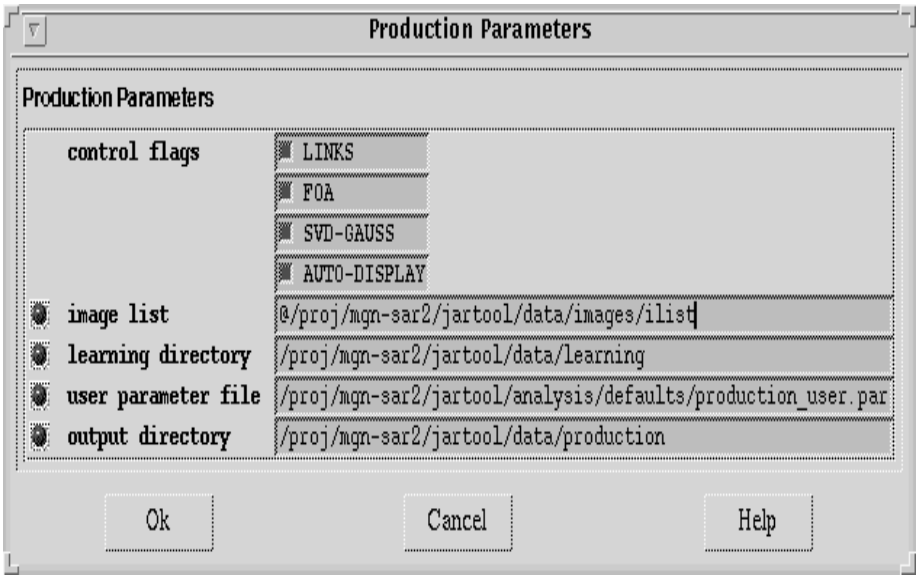


Figure 2.2: The parameter input GUI (PIG) for the JARtool production process.

After specifying all the required parameters to the data entry widgets, click the *OK* button to begin processing. If you instead change your mind and decide you don't want to do a function, just click on the *Cancel* button. Most PIG's come with hypertext help files, which can be accessed through the PIG's *Help* button.

Another feature that is common to many PIG's is a builder button which appears as a red ball at the left of certain parameter input widgets. The builder buttons

provide the user with an alternative means for entering parameters (instead of just typing in the values).

Some parameters are optional. These are designated by square brackets around the parameter name. For most PIG's the *OK* button will be disabled until all of the required parameters have been specified by the user. A nice feature of the PIG's is that they have memory. If you invoke the same function more than once, the PIG will initialize the parameters to the last set of values used with the function. The PIG's have factory default values that are used initially, but the user can customize these for their own system.

2.14 Entry Boxes

Many of the parameter input GUI's (PIGS) and the JARtool dialog boxes have *entry boxes*, pink rectangular areas for entering text. Provided that an entry box has focus, the user can simply type on the keyboard and the text will appear in the box. To give a particular box "focus", just click in the box with the left mouse button. You will see the insertion cursor appear in the box indicating where it is expecting your input.

Entry boxes typically have the following keyboard bindings:

CTRL-a Move the insertion cursor to the beginning of the box.

CTRL-e Move the insertion cursor to the end of the box.

CTRL-k Delete the text from the insertion cursor to the end of the box.

BACKSPACE Delete the character behind the insertion cursor.

DELETE Same as backspace. Both of these keys will also delete selected (highlighted) text.

CTRL-d Delete the character in front of the insertion cursor.

CTRL-f Move the insertion cursor forward one character.

CTRL-b Move the insertion cursor backward one character.

TAB In entry boxes that expect a filename or lists of filenames, the tab key provides filename completion. In other words, you can type the first part of the filename, hit tab, and the system will automatically try to complete the filename.

In addition, entry boxes have the following mouse bindings:

Clicking the left mouse button will place the insertion cursor at the position of the mouse click.

Dragging the left mouse button will select (highlight) text.

Dragging the middle mouse button will "scan" or "scroll" text that is too long to fit within the entry box.

2.15 Builder Buttons

Several parameter inputs GUI's (PIGs) contain a red button next to some of the parameters. These buttons provide an alternative means for entering parameter values (as opposed to just typing the values into the entry box). Clicking the left mouse button on the builder button will bring up a menu of builder options. For example, if a list of images is needed as a parameter, clicking on the builder button will bring up a menu with options to (1) build a list with the *XListEditor* graphical interface or (2) build a list with the *List Expression* tool, or (3) invoke a file browser to look for a list that was previously stored on the file system.

2.16 File Browser/File Selection Box (FSB)

The *File Browser/File Selection Box (FSB)* provides a convenient interface for the user to browse through the file system and identify filenames or directories for use as PIG or dialog box inputs. The user's selection appears in the entry box area labeled "Current Selection". Clicking the mouse on the *Ok* button near the bottom of the interface (or hitting a carriage return while in the entry area) will send the current selection back to the calling program and terminate the FSB.

Physical Layout

The FSB consists of several components arranged in the following way. Near the top of the interface are two dark-blue buttons labeled "Hot Directories" and "Templates". Below these buttons are two entry box areas labeled "Current selection" and "Current template". The main body of the interface is a set of three list boxes. The list boxes show the hierarchical directory structure associated with the current selection. Below the list boxes are three buttons used to return from the FSB, quit, or get help documentation: *Ok*, *Cancel*, and *Help*.

Hot Directories

Clicking and holding the left mouse button on *Hot Directories* will bring up a pull-down menu of "hot directories". Releasing the mouse button over one of these items will place it in the current selection and update the listboxes accordingly. Releasing the mouse button over the "Add Entry" menu option will add the directory from the current selection entry box to the list of hot directories. If the current selection refers to a file rather than a directory, the directory path to the file will be stripped off and added to the list.

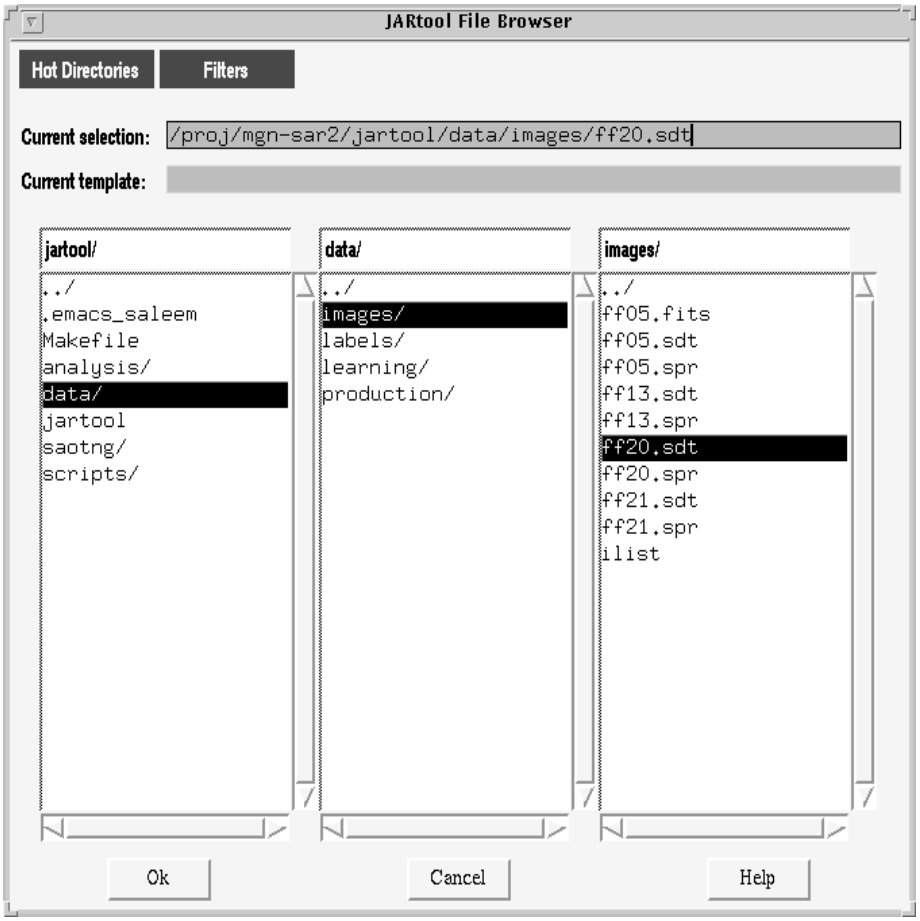


Figure 2.3: JARtool file browser/file selection box.

Programmers Note: The file containing the set of hot directories can be specified through a command line flag (**-dirs**) when the FSB is invoked. If this flag is omitted, the FSB checks the current working directory for a file named user.directories. If that file doesn't exist, the FSB will look in a directory defined by the environment variable UPARM for a user.directories file. In addition to the hot directories defined by the file, the list also includes automatically the current working directory.

Templates

The *Templates* button is similar to the *Hot Directories* button. By releasing over an item in the template menu, the item becomes the current template. A template in the "Current template" entry box can be added to the menu by releasing over the "Add template" menu option. Note that in addition to the predefined templates, the list also automatically includes the "*" template, which matches any file.

Programmers Note: The file containing the set of predefined templates appearing in the menu can be specified through a command line flag (**-tmps**) when the FSB is invoked. If this flag is omitted, the FSB checks the current working directory for a file named user.templates. If that file doesn't exist, the FSB will look in a directory defined by the environment variable UPARM for a user.templates file.

Current Selection

This entry box holds the user's current selection. The selection can consist of a complete directory pathname and/or a partial/full filename. The current selection can be set and modified in several ways:

Command Line Argument The initial selection can be specified through a command line argument when the FSB is invoked using the flag **-isel**. (This option is only relevant to programmers and users who are calling individual JARtool components from the UNIX command line.)

Typing The user can use the mouse to position the cursor within the entry box and then enter characters from the keyboard. The **Backspace** key deletes characters behind the cursor, while the **CTRL-d** key deletes characters in front of the cursor. Special editing features similar to those used in emacs are also available.

Clicking Clicking the mouse on any item in the list boxes will place that item (along with its associated path) into the current selection and update the listbox display accordingly.

Hot Directories Releasing the mouse over an item in the hot directories menu will make that item the current selection.

Tab Completion If the cursor is at the end of the text in the entry area, the user may hit the Tab key (on the keyboard) to complete the current segment of the pathname. If there are no matching items or if no additional information is available from Tab Completion, the interface will beep. If there are multiple matching items, the greatest common prefix will be completed.

Current Template

The current template acts as a filter on all filenames displayed in the list box areas. Only filenames that match the form of the template will be displayed. To allow updating when the user types, all templates have an implied * on the right end. Thus, the template *.s will match the files cat.s, dog.s, as well as cat.sdt, dog.spr, etc. The template acts only on files, not on directories. The template "*" matches with any file.

Setting the current template can be done through a command line flag (**-itmp** at startup, by typing in the entry area, or by using the *Templates* menu. Note that because most templates contain the special asterisk character (*), you must enclose the template in double quotes if using the command line option, e.g., **-itmp "*.sdt"**.

One word of caution: since only files that match the template are displayed in the listboxes, it may seem that some of your files are "missing". Of course, this is only in the listbox display; your files are still on disk. Changing the template to "*" will enable you to see them again.

List Boxes

The three list boxes are the "guts" of the FSB. These show the hierarchical directory structure associated with the "Current Selection". The label at the top of each list box shows the directory whose contents are displayed in the list box. For example, in Figure 2.3 the middle list box has the label $\mathrm{data}/$, hence, the items appearing in this list box are the contents of the directory

$/\mathrm{proj}/\mathrm{mgn\text{-}sar2}/\mathrm{jartool}/\mathrm{data}/$

The item ../ refers to the parent directory. Since $\mathrm{images}/$ is highlighted in the middle list box, the contents of this subdirectory of $\mathrm{data}/$ are displayed in the right list box.

Ok

Clicking the mouse on the *Ok* button will cause the FSB to terminate and return the current selection to the calling program. Hitting the Return key while in the current selection entry box performs the same function.

Cancel

Clicking the mouse on the *Cancel* button will cause the FSB to terminate and return a null selection ("") to the calling program.

Help

Clicking the mouse on the *Help* button will cause a hypertext help file to be displayed.

2.17 List Files

List files are simply ascii files with one item per line. For example, the following items (the names of four image files) could stored in a list file called old4.

/home/venus/ff05.sdt
/home/venus/ff13.sdt
/home/venus/ff20.sdt
/home/venus/ff21.sdt

List files can be used whenever a parameter expects a list of items. The @ symbol must be placed infront of the list filename so that JARtool knows to get the filenames from within the list file. This convention is called @-notation. Thus, if an entry box asks for a list of images, the following specifications are equivalent:

1. /home/venus/ff05.sdt /home/venus/ff13.sdt /home/venus/ff20.sdt... /home/venus/ff21.sdt
2. @old4

The list file method of specification can be mixed with explicit filenames as in the following example:

- @old4 /home/venus/ff32.sdt

Note that this specification contains five images: the four listed in old4 plus ff32.sdt.

2.18 Busy Boxes

Several functions in JARtool such as learning can take some time to complete. To let the user know that these processes are runnning and to provide a mechanism for aborting a running process, the JARtool system starts up a busy box for selected (time-consuming) functions. The busy box will appear as a jar that is continuously filling with water. Unfortunately, the fill rate is not proportional to the progress of the function, so a jar may fill and empty many times before a function is completed.

The user can abort a function by clicking on the *Quit* button of the appropriate busy box. This action will kill the function and any background subprocesses it may have spawned.

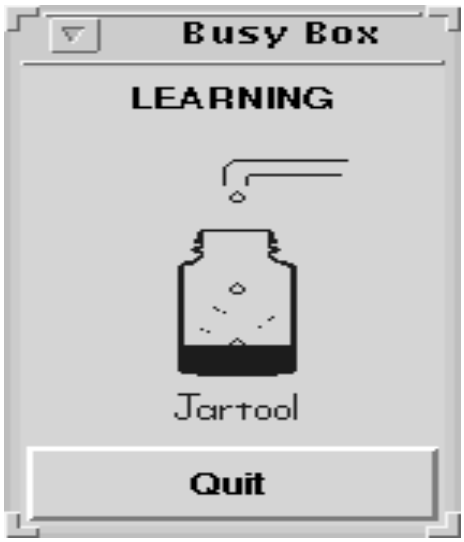


Figure 2.4: Busy box for learning.

Warning: There is a potential bug with the busy box kill button. When clicked, the busy box does a ps -l system command to list currently running processes and then it tracks down all children of the primary process. However, in the time between performing the ps command and the time when the child processes are killed, it is possible that new processes could have been spawned. These new processes will not be killed.

Programmers Note: Another potential hazard is the fact that the *child_processes* procedure is based in part on the format of the UNIX ps -l output as it appears on SUN workstations. This dependency could lead to problems if JARtool is ported to other platforms.

2.19 Marker Analysis Tasks

SAOtng now supports marker analysis tasks. Click the right mouse button inside or on the boundary of a marker and drag down to the "Marker Analysis Tasks" menu option. This action will bring up a submenu with a variety of functions. The available options are dependent upon the marker type. For example, with line markers it is possible to plot the intensity profile along the line. With box markers, you can find statistics of the pixels inside the box. Users can also add their own marker analysis routines by following the examples in the manalysis.cmds file (under the

JARtool directory structure $\$\mathrm{jartool/saotng/cmds/}$).

Chapter 3

Images

The *Images* menu button provides options for loading, saving, and printing images. To load an image, click the left mouse button and pulldown the options menu. Release the button over the *Load Image* option and type the image filename into the entry box. Instead of typing into the entry box, you can click the left mouse button on the builder button to the left of the entry box. This will invoke a graphical file browser that can be used to select the filename. After hitting *OK*, the selected filename will be loaded into the image area.

The preferred JARtool image format is VIEW format; however, the native image format of SAOtng is FITS (Flexible Image Transport System), a format which was developed within the astronomy community. FITS format images can be read and displayed in JARtool; however, the learning and production operations as well as some of the tools only work with VIEW format. There is a format conversion operation under the *Tools* menu bar button that can be used to convert a number of formats such as FITS, RAS, PDS, and VICAR into the JARtool VIEW format. Details of the VIEW format are covered in Appendix B.

Two parameters in the start-up configuartion file control the maximum size image that can be loaded at full resolution. Larger images will automatically be block-averaged to reduce the size. Do not set these parameters to be too large, however, as you will quickly consume all of your system's memory. The default values are

$$\begin{aligned} &\mathrm{XIMTOOL\_FRAME\_WIDTH} \;=\; 1280 \\ &\mathrm{XIMTOOL\_FRAME\_HEIGHT} \;=\; 1280 \end{aligned}$$

Note that there is no support for true color images. Indexed color images, in which each pixel is an index into a color table, are not supported either unless the

color table happens to be one of the default tables provided by SAOtng.

Chapter 4

Frames

A frame is simply a buffer containing an image. Multiple frames may exist at one time, but only one of these is mapped to the image display area at a time. The *Frames* menu button provides options for creating new frames and selecting which frame will be displayed in the image area. As an example, the user may load an image into frame 1, create a new frame (frame 2), and load a new image there. The user may then go back to the first image by selecting the "Go to Frame 1" menu option.

The user can blink (flip the display back and forth) between image frames using the *Blink* accelerator button or the blink option under the *Control Panel* accelerator button.

Note: Each frame consumes system memory so be careful not to create too many frames. This advice is especially relevant when using JARtool features that automatically allocate frames (e.g. the *Overlay* tool or *Production* with *Auto-Display* enabled).

Chapter 5

Labeling

The *Labeling* menu button provides a method for the user to generate training data by circling (or otherwise marking) objects of interest in the imagery. To label an image, simply choose the desired marker shape from the "Set Default Marker Shape" option in the *Labeling* menu. Note that the interface is always in labeling mode. The menu button simply provides control over aspects of the process such as the marker shape and color, as well as providing functions to load and save markers.

Assuming that you have selected the circle marker shape, click on the image with the left mouse button, drag, and release. The initial click sets the center of a circle, while the drag adjusts the size. Releasing the button sets the final size of the circle.

Markers can be easily moved around, resized, and deleted.

Move – To move a marker, click the left mouse button on or inside the marker and drag to the desired position.

Resize – To resize, click the left mouse button on the edge of the marker and drag to the desired size. Note that the cursor changes shape when inside a marker or on top of the marker edge.

Delete – To delete a marker, hit the delete or backspace key while the cursor is inside or over it. (There is an undelete option under the *Labeling* menu.)

Numerical identification labels and text can be attached to any marker. Simply move the cursor over the marker edge, click and hold the right mouse button to bring up a pulldown menu. Under the "Set Type" option there will be predefined labels that the user can select. Simply release the right mouse button over the appropriate label. To change a label, repeat the procedure; the old label will be overwritten. Note that these labels are defined in a file that the user can customize

($jartool/saotng/cmds/markmap.cmds). Avoid using the label 0 as this label has a special meaning to some of the JARtool programs.

In addition, it is possible to attach more general text such as "Best volcano I have ever seen". To do this, release the right mouse button over the "Attach/Edit Text" option. This will bring up an entry box into which you can type text. Multi-line text is allowed. Click the *OK* button when finished.

To save the markers to a file, select the "Save Markers" option under the *Labeling* menu button. There is also an option to load markers from file. Note that markers are saved in a file format called JTRI, which is a JARtool extension of the basic SAOtng marker format. Appendix C contains more details on the JTRI file format.

Chapter 6

Database

The *Database* menu button is used to invoke the *Query Tool*, a graphical user interface for interacting with the JARtool database. (Before using functions in this chapter, the database must be up and running. Appendix A contains the database installation instructions.) With the *Query Tool* the user can

- Insert and update information in the database.
- Retrieve information from the database using SQL queries.
- Save the retrieved information.
- Visualize retrieved information.

Information in the database is organized into structures called tables. Each table has a number of attributes. One can think of the attributes as column headings. For example, in an employee records database, we might want to store an employee ID number, employee name, and date of hire. These would be the attributes and each employee would be represented by one row in the table.

Now suppose we wanted to also keep track of the employees' hometown. There are several ways this could be done. We could simply add to the table a column containing each employee's hometown as a text string (e.g., "Pasadena"). However, this is somewhat inefficient for storage and indexing, since the same hometowns would be repeated frequently. An alternative is to make a second table containing a hometown ID number and a hometown string. To the first table, we would add a column containing *Hometown_ID_Number*.

When we look at the employee records, however, we do not want to see a *Hometown_ID_number*. Instead, we want to see the hometown as ordinary text. We can achieve this effect by defining a *view*, which is a "virtual table" derived from the underlying tables. The hometown ID numbers would be converted into regular text

by looking at the second table (joining) to decode the ID number. The lower level tables help the database work efficiently, but the views present the information in a more user-friendly way. In the JARtool database, the views will be of primary interest although interaction with the low-level tables may also be necessary.

To provide easy access to the information in the database, the *Query Tool* interface was designed to be similar in style and layout to the main JARtool interface. There are eight components as shown in Figure 6.1. Clockwise from the left we have:

1. Accelerator Buttons
2. Menu Bar
3. Results Window
4. Results Buttons
5. Query Window
6. Query Buttons
7. Help Button
8. Accelerator Button Popup Menu

6.1 Accelerator Buttons

The buttons along the left side of the display are called accelerator buttons. The function of each button in the standard set is described here. (We use the qualification "standard set" because the buttons can be re-defined by the user.)

Tables – list the tables in the active database.

Views – list the views in the active database.

Attr(Table) – list the attributes (column headings) of the specified table.

List(Table) – list the records (rows) of the specified table.

Img(Obj) – list the images which contain the specified object.

Img(Lat, Lon) – list the images which contain the specified point on the planet.

ImgRgn(Auth) – list the images and the region files produced by a specified author (or set of authors).

RgnAuth(Img) – list the authors who labeled the specified image and the names of the region files produced.

Obj(Img) – list all objects from the specified image.

Obj(Lat, Lon) – list all objects in a specified area of the planet.

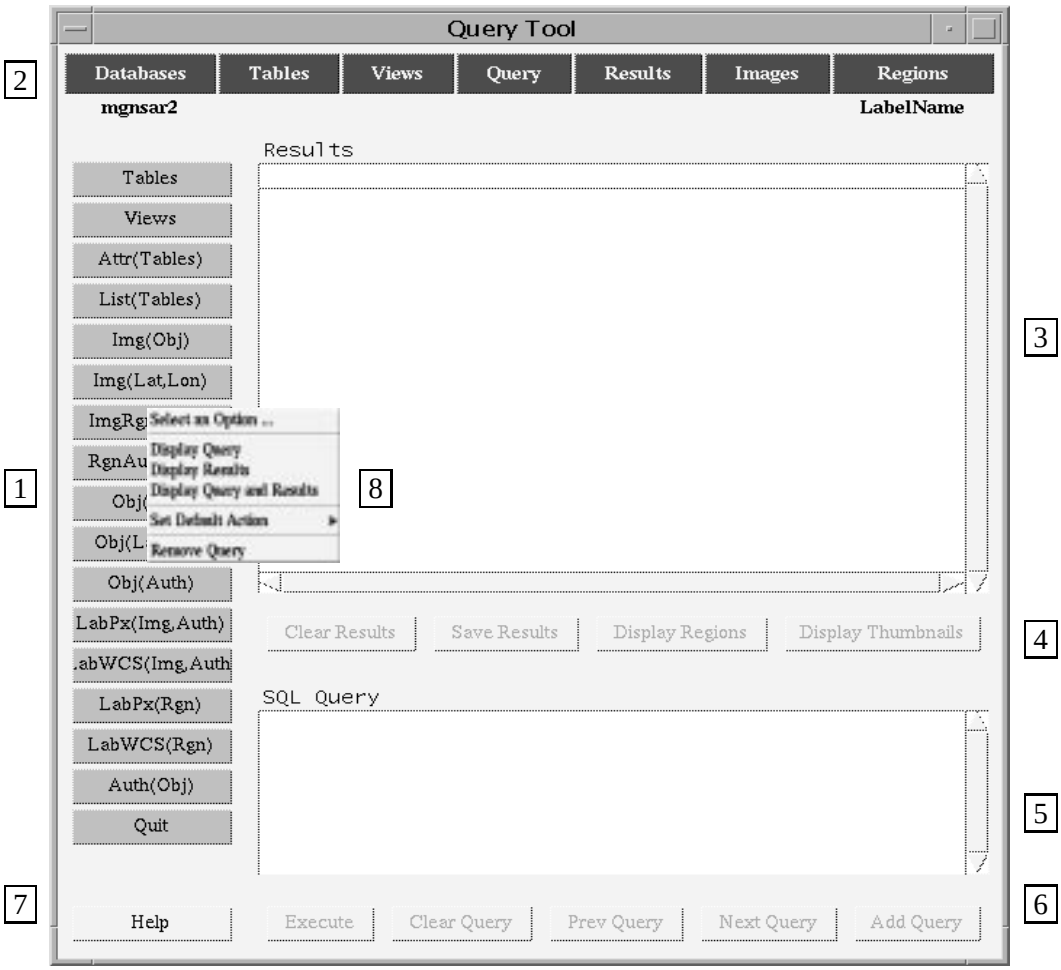


Figure 6.1: The Query Tool graphical user interface.

Obj(Auth) – list all the objects labeled by the specified author.

LabPx(Img, Auth) – list all labels in pixel coordinates that were produced for the specified image by the specified author.

LabWCS(Img, Auth) – list all the labels in wcs (world coordinate system) coordinates that were produced for the specified image by the specified author.

LabPx(Rgn) – list all the labels in pixel coordinates from the specified region file.

LabWCS(Rgn) – list all the labels in wcs coordinates from the specified region file.

Auth(Obj) – list all authors who labeled the specified object.
Quit – exit from the Query Tool.

To activate a particular accelerator button, simply position the cursor over the button and click the left mouse button. Except for the *Quit* function, clicking the left mouse button over an accelerator button will cause one of the following actions:

- If no additional information is needed from the user, the query will be performed and the results displayed in the Results Window.
- If additional information is needed from the user, the SQL query will be displayed in the Query Window. The user can then enter the required parameter values (for example, a latitude and longitude) by editing the arguments of the #define statements at the top of the query and then execute the query.

6.2 Menu Bar

The menu bar consists of the buttons near the top of the display. Clicking the left mouse button on a menu bar item causes a pulldown menu to be displayed. To select an option from the pulldown menu, move the mouse over the option and click the left mouse button. (An option can alternatively be selected by dragging and releasing the left mouse button.)

Databases

This menu contains the following options:

Connect to . . . – connect to a database server.
Insert Labels – insert labeling records into the database. You will be asked to provide the name of a file. The file will contain a number of lines, and each line will have an image filename, a jtri filename, and an author name separated by spaces. The appropriate tables will be updated in the database.
Insert Images – insert images into the database. You will be asked to provide the name of a file. The file will contain a number of lines, and each line will have an image filename, minimum latitude, maximum latitude, minimum longitude, maximum longitude, image height, image width, and look direction (all entries separated by spaces). The appropriate tables will be updated in the database.

Clean Database – remove all records from the entire database (proceed with caution).
Disconnect – disconnect from a database server.
Quit – exit from the *Query Tool*.

Tables

This menu contains the following options:

List Tables –list all the tables in the active database.
List Attributes – list the attributes (column headings) of a particular table.
List Records – list the records (rows) of a particular table.
Clean Objects Table – delete all object related information from the database.
Update Objects Table – re-create the objects table and update all other tables which have object related attributes.

Views Menu

This menu contains the following options:

List Views – list all the views in the active database.
List Attributes – list the attributes (column headings) of a particular view.
List Records – list the records (rows) of a particular view.

Query Menu

This menu contains the following options:

Clear Query – clear the Query Window.
Prev Query – display the previous query in the Query Window.
Next Query – display the next query in the Query Window.
Add Query – add the currently displayed query to the set of accelerator buttons.
Remove Query – remove an accelerator button.
Execute Query – execute the query displayed in the Query Window.
Execute Queries from File – execute queries stored in a text file.

Results Menu

This menu contains the following options:

Clear Results – clear the results window.

Save Results (as text file) – save the results as a text file.

Display Images and Regions – if *ImageName* and *RegionName* are among the displayed attributes, selecting this option will display in successive frames each image with the corresponding region file overlaid.

Images Menu

This menu contains the following options:

Display Images – if the displayed results contain *ImageNames*, this option will display the images in successive frames.

Display Thumbnails – if the displayed results contain *ImageNames*, this option will display a single image of thumbnails, which are reduced size versions of the original images.

Save Results (as image list) – if the displayed results contain ImageNames, this option will produce a text file containing the image filenames. This feature is useful since the learning and production routines operate on lists of image files.

Regions Menu

This menu contains the following options:

Set Region Color – set the color in which labels will be displayed in JARtool.

Display Regions – if labels are displayed in the Results Window and an image is loaded into JARtool, this will overlay the labels on the displayed image.

Save Results (as RGN file) – if labels are displayed in the Results Window, this will save them to a RGN format file.

Save Results (as JTRI file) – if labels are displayed in the Results Window, this will save them to a JTRI format file.

Save Results (as JTRI list) – if JTRI files are displayed in the Results Window, this will produce a text file containing a list of the JTRI files. This feature is useful since the learning routines and many of the tools operate on lists of JTRI files.

6.3 Results Window

This window is used by the Query Tool to display the results of user performed queries. If the results are too large (too wide or too many results), scroll bars will appear, which can be used to bring more information into the display area.

6.4 Results Buttons

This part of the interface contains some results-related buttons. The buttons and the functions they perform are as follows:

Clear Results – clear the Results Window.

Save Results – save the displayed results to a text file.

Display Regions – if labels are displayed in the Results Window and an image is loaded into JARtool, this button will display all the labels that lie on the displayed image.

Display Thumbnails – if a set of image names is displayed in the Results Window, this button will display a thumbnail image containing reduced size versions of all the images.

6.5 Query Window

This window can be used to enter SQL queries. (We do not attempt to explain the SQL language, but there are many books available on the topic. You can learn some of the basic syntax by examining the pre-canned queries, which are mapped onto the accelerator buttons.) To enter an SQL query, position the mouse over the Query Window and click the left or middle mouse button to give the window focus. Then simply type in your query. To execute the query, click the *Execute* button or select "Execute Query" from the *Query* menu.)

6.6 Query Buttons

This part of the interface contains some query related buttons. The buttons and the functions they perform are as follows:

Execute – execute the query displayed in the Query Window.

Clear Query – clear the Query Window.

Prev Query – display the previous query in the Query Window.

Next Query – display the next query in the Query Window.

Add Query – add the query displayed in the Query Window to the Accelerator Buttons.

6.7 Help Button

At the lower left of the display is a *Help* button that provides access to the hypertext documentation. Click the left mouse button on any highlighted words to get more information.

6.8 Accelerator Button Popup Menu

The accelerator button popup menu contains options to control the behavior of the accelerator button. There are three options: (1) display the query in the Query window, (2) execute the query and display results without displaying the query itself, and (3) display both the query and the results. To activate this menu, move the cursor over an accelerator button and hold down the right mouse button. To select an option, drag the mouse over the desired option and release. The available options are:

Display Query – display in the Query Window the query associated with the selected accelerator button.

Display Results – execute the query associated with the selected accelerator button and display the results in the Results Window.

Display Query and Results – display (in the Query Window) the query associated with the selected accelerator button, execute the query, and display the results in the Results Window.

Set Default Action – change the default action of the selected accelerator button to one of the above.

Remove Query – remove the accelerator button.

Chapter 7

Learning

The *Learning* menu button provides access to the JARtool learning capability, which can be used to build appearance models for objects of interest given training examples. In this chapter we will provide a brief overview of the learning algorithm and then focus on how to use the software. For more details on the algorithm, consult the references listed in the bibliography.

7.1 Algorithm Overview

The learning algorithm consists of three components: focus of attention (FOA), feature learning, and classification learning.

The FOA is intended as a quick pre-filter to eliminate areas of an image that clearly do not have any volcanoes. The output of the FOA is a discrete set of candidate locations, which the algorithm has deemed to be worth examining in more detail.

The feature learning component uses principal components analysis to infer features from statistical regularities in the training examples. The types of features that can be learned are restricted to linear combinations of the pixel data. Thus, each "feature" can be viewed as a 2-dimensional template or mask. The inner product (correlation) of the template with a region will be defined as the value of the feature.

The classification learning component is used to determine the mapping from vectors of feature values to class identity (*volcano* or *not-volcano*). Classification learning is accomplished using a standard 2-class, supervised technique. The algorithm uses labeled feature vectors from both the *volcano* and *not-volcano* classes. The *not-volcano* examples consist of any objects that passed the FOA test, but weren't

designated by the geologists as possible volcanoes.

7.2 Running the Learning Software

From the *Learning* menu button, select the "LEARNING" option. This action will bring up a PIG (parameter input GUI). Training examples are specified to the system by giving a list of images and the corresponding list of jtri files. Each jtri file contains markers (circles) that indicate the location of objects of interest.

Inputs:

control flags – These flags allow selected portions of the learning algorithm to be turned on or off. However, since the processing is sequential, it does not make sense to turn off early stages unless they were performed during a previous session. All flags should normally be set to on (this is the default). The **Links** flag creates the learning output directory structure and establishes soft links to the images and label files. The **FOA** flag causes the algorithm to learn an FOA filter and apply the result to the training images (with scoring). The **SVD-Gauss** flag causes the algorithm to do feature learning and classification using singular value decomposition (SVD) and a Gaussian classifier.

image list – This parameter specifies the set of images (in VIEW format) to be used for training. The user may simply type the filenames (separated by spaces) into the entry box. Pathnames are required unless the images are in the current working directory. The user may also specify a list file using @-notation. Lists may contain a combination of @-notation and explicit filenames. The red builder button next to the entry can be used to construct a list using a graphical interface or to look for a specific list that was previously stored in a file.

jtri list – This parameter specifies the set of ground truth files (in JTRI format) to be used for training. It is important that these files exactly correspond with the set of images specified in **image list**.

user parameter file – This entry should be the name of a file that sets up various user-defined parameters that affect the learning process, e.g., the size of the ROI's, the threshold applied to the FOA output, etc. The user can modify the parameters by copying the default file and changing it with a text editor such

as emacs. Changing this file is only recommended for experienced users. Refer to Appendix D for more information.

output directory – This entry specifies the root directory that will contain the results of learning, i.e., the learned models. The directory and its subdirectory structure will be created if it does not already exist. If the directory and subdirectory structure already exist, the contents may be overwritten.

Outputs:

The primary outputs of learning are the filter to be used for the FOA algorithm, the set of templates (principal components) to be used to extract features, and the statistics used by the classifier to map from feature vectors to class labels.

The outputs are organized into several subdirectories: images, labels, pars, FOA, and CLASS. The images and labels directories contain symbolic links to the actual images and labels. The pars directory saves all the user and system parameters used during the learning process.

The FOA directory contains the matched filter as well as its performance assessment on the training set. If you are experimenting with the system on a new data set, you may want to look at the $.\mathrm{sc}$ files to evaluate whether the FOA threshold you are using is reasonable. The first two entries in the $.\mathrm{sc}$ files are the number of false alarms and the number of true detections, respectively (assuming all label categories are treated as "object"). The next n entries are the number of detections in each label category, where n is the number of label categories. The last n entries are the number of ground truth objects in each label category. From these numbers you can get a rough idea of how the FOA is performing at the selected threshold. Keep in mind that the FOA is designed to be aggressive (low miss-rate, but potentially high false alarm rate). See Appendix D for more information on how to set the FOA threshold.

The CLASS directory contains several subdirectories corresponding to different numbers of features. The subdirectory with the largest number of components contains the templates (principal components) used to extract features. Each of the $\#\_\mathrm{components}$ subdirectories contains classifier statistics for the specified number of features.

Chapter 8

Production

The *Production* menu button provides access to the JARtool production capability, which is used to apply an appearance model derived during learning to a new set of images with the goal of locating novel instances of the object of interest. In this chapter we will provide a brief overview of the production algorithm and then focus on how to use the software. For more details on the algorithm, consult the references listed in the bibliography.

8.1 Algorithm Overview

The production algorithm consists of three components: focus of attention (FOA), feature measurement, and classification.

The FOA uses the matched filter derived during learning as a pre-filter to screen local regions of each image. Locations where the filter response exceeds a threshold are clustered together (nearby hits are attributed to the same object). The cluster centers are passed on to the feature measurement stage.

Feature measurement is performed on a region of interest (ROI) around each cluster center. However, the ROI is first normalized for variations in contrast level. Feature values are measured by computing the inner product of the normalized ROI with the principal components templates obtained during learning. Thus, each ROI is converted from a high-dimensional vector of pixel values to a low-dimensional vector of feature values.

For each ROI, the classifier assigns a class (a posteriori) probability based on the associated vector of feature values. The probability values are thresholded and any ROI's with posterior probabilities exceeding the threshold are declared to be volcanoes.

8.2 Running the Production Software

From the *Production* menu button, select the "PRODUCTION" option. This action will bring up a PIG (parameter input GUI) with the following inputs.

Inputs:

control flags – These flags allow selected portions of the algorithm to be turned on or off. Since the processing is sequential, it does not make sense to turn off early stages unless they have been performed in a previous session. All flags should normally be set to on (this is the default), with the possible exception of **Auto-Display** and **Auto-Insert**. The **Links** flag creates the output directory structure and establishes soft links to the image files. The **FOA** flag causes the algorithm to generate candidate locations by applying an FOA filter to the images. The **SVD-Gauss** flag causes the algorithm to do feature measurement and classification using a Gaussian classifier. The **Auto-Display** flag causes the images and corresponding jtri files determined by the production algorithm to be displayed in successive frames at the completion of production. For large jobs, you will probably want to turn this flag off. The **Auto-Insert** flag causes the results of the production run to automatically be inserted into the database. If you initially run production with this flag set to off and later decide to insert the results, you will need to use the "Insert Labels" option under the *Query Tool Databases* menu. The production routine generates a file called $\mathrm{class.dbs}$ in the $\mathrm{database}$ subdirectory under the production output directory. This file is needed by the "Insert Labels" routine.

image list – This parameter specifies the set of images (in VIEW format) to be used for testing. The user may simply type the filenames (separated by spaces) into the entry box. Pathnames are required unless the images are in the current working directory. The user may also specify a list file using @-notation. Lists may contain a combination of @-notation and explicit filenames. The red builder button next to the entry can be used to construct a list using a graphical interface or to look for a specific list that was previously stored in a file.

learning directory – This parameter indicates the root directory where the model parameters generated during learning were stored, i.e. the **output directory** specified during the learning process.

user parameter file – This entry should be the name of a file that sets up various user-defined parameters that affect the production process, e.g., the size of the ROI's, the threshold applied to the FOA output, etc. The user can modify the parameters by copying the default file and changing it with a text editor such as emacs. Changing this file is only recommended for experienced users. Refer to Appendix E for more information.

output directory – This entry specifies the root directory that will contain the results of production, i.e., the jtri files containing the detected object locations. This directory and its subdirectory structure will be created if it does not already exist. If the directory and subdirectory structure already exist, the contents may be overwritten.

Outputs:

The output of production is a set of jtri files containing the coordinates of each detected object. If the **Auto-Display** flag is on, the results will be displayed in consecutive frames. If the **Auto-Insert** flag is on, the results will be inserted into the appropriate tables of the database.

Chapter 9

Tools

9.1 XListEditor

Many components of JARtool are structured to work on lists of images or lists of jtri files. The *XListEditor* enables a user to construct lists using an interface similar to the *File Browser/ File Selection Box*. Lists can be saved in files and then supplied to programs using @-notation. This approach is often more convenient than typing all the image or jtri names into an entry box.

The *XListEditor* contains three listboxes that allow the user to browse around through the directory structure. In addition, there is a fourth list box that is separated slightly from the others. This box maintains a list of the items that have been selected by the user. To add an item to the current list, double-click on the item in one of the first three listboxes. To remove an item, double-click on it in the fourth listbox.

On the *XlistEditor* under the *File* pulldown menu button, there are options for saving the current list to file, loading a list from file, and clearing the current list.

9.2 List Expression

The *List Expression* tool allows the user to specify a list of files having names like image1.sdt, image2.sdt, image3.sdt, etc. The user provides a base filename, a set of frame numbers, and a filename suffix. The frame numbers may be a comma or space separated list. (Note: the use of the word frame in this context is unrelated to the *Frames* discussed in Chapter 4.) Colon (:) notation, i.e., start_frame:increment:end_frame, is also supported. For example, 0:5:20 would specify frames 0,5,10,15, and 20. The increment may be omitted, in which case the de-

Figure 9.1: The XListEditor provides a graphical interface for constructing lists, which are used by many of the JARtool functions.

fault value is 1. Negative increments are permitted.

If the start_frame is given as 001 or some similar fixed-width expression with leading zeros and the end_frame is of the same width, then leading zeros will be used to maintain fixed width frame numbers.

For each frame number specified, a filename is generated by concatenating the base filename, the frame number, and the filename suffix. For example, if the base filename is /home/venus/image, the frame numbers 1:3, and the suffix .sdt, then the *List Expression* tool would generate:

home/venus/image1.sdt

home/venus/image2.sdt
home/venus/image3.sdt

9.3 Format Conversion

The *Format Conversion* tool enables the user to convert several popular image formats, including VICAR, FITS, RAS, RAS24, and PDS to the VIEW format required by JARtool.

Inputs:

input image list – This parameter specifies the set of images to be converted. The user can type the filenames (separated by spaces) into the entry box. Pathnames are required unless the images are in the current working directory. The user can alternatively specify a list file using @-notation. Lists can contain a combination of @-notation and explicit filenames. The red builder button next to the entry can be used to construct or specify a list using one of several methods.

The format of each image in the input list is determined from the filename extension. Currently supported input formats include VICAR (.vic), FITS (.fits), RAS (.ras), RAS24 (.ras), and PDS (.img). Note that with SUN rasterfile (RAS and RAS24) formats, the converted images are greyscale even if the original images were indexed color or RGB.

output image list – This parameter specifies where to put the VIEW format output images and how to name them. The **output image list** can be a list of filenames, the name of a directory, or empty. If empty, the images are written in the current directory using their original base names, but with the proper VIEW filename extensions.

Outputs:

The output consists of a set of VIEW format images, which will be located on the file system according to the **output image list** parameter.

9.4 JTRI Filter

The *JTRI Filter* tool takes as input a list of jtri files and outputs a list of jtri files. Each jtri entry is filtered according to criteria specified by the user. For example, a

typical request might be: show me only those jtri entries which the scientists labeled as category 1 and which have radius larger than 20 pixels. The filter for this example would have the following form:

$$(L == 1)\ \&\ (R > 20.0)$$

The builder button for the filter entry generates two menus in a calculator format containing the allowed attributes that can be filtered and the allowed operations. General algebraic expressions can be used as the basis of a filter operation. For example,

$$(SQRT(H * W) > 20.0)$$

where H is height and W is width.

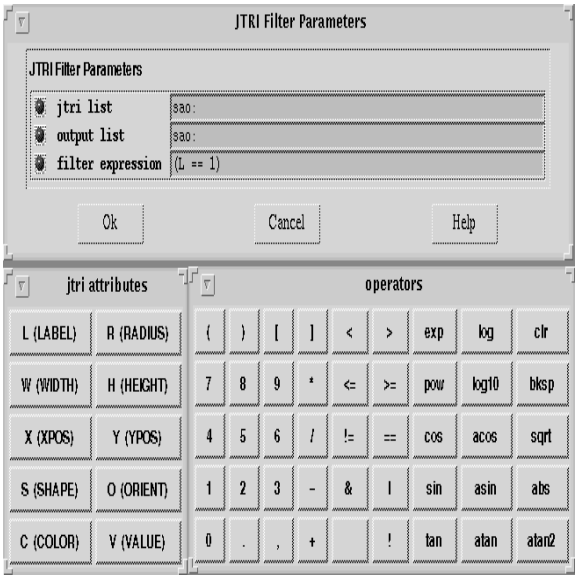


Figure 9.2: PIG for the JTRI filter tool. The expression builder button brings up two calculator-style menus that can be used to specify a filter expression.

Hints:

- The filename **sao:** has a special meaning. In particular, it refers to the current display. Thus, if **sao:** is specified as the input, the currently displayed

markers will be filtered. Similarly, if **sao:** is specified as the output, the filtered markers will be drawn on the current display (old markers will first be deleted).

- The JTRI filter tool can be used to copy a list of jtri files to a new set of names by specifying the filter expression "1".
- The JTRI filter tool can also be used to filter the output of production so that only those regions identified as instances of the target object are actually displayed. In other words, regions identified by the FOA but rejected by the classifier are not displayed. To acheive this effect, use the filter expression:

$$(L == 1)$$

9.5 Confusion

The *Confusion* tool enables the user to compare two sets of jtri files and calculate confusion matrices. This feature is useful both for comparing the consistency of one labeler versus another and for comparing algorithms against "ground truth".

Inputs:

jtri list (detections) – This parameter specifies one set of jtri files. If the two sets of jtri files correspond to "detections" and "ground truth", then this first list should be the "detections". The user may simply type the filenames separated by spaces into the entry box. Pathnames are required unless the images are in the current working directory. The user may also specify a list file using @-notation. Lists can contain a combination of @-notation and explicit filenames. The red builder button next to the entry can be used to construct or select a list using one of several available methods.

label limits (detections) – This input consists of two parameters specifying the lower and upper limits of labels that can appear in the "detection" jtri files.

jtri list (ground truth) – This parameter specifies the second set of jtri files. If one set of the jtri files corresponds to "ground truth" it should be specified here.

label limits (ground truth) – This input consists of two parameters specifying the lower and upper limits of labels that can appear in the "ground truth" jtri files.

scoring distance threshold – A detection and ground truth entry are matched up if they are within this distance from each other.

border exclusion threshold – Detections and ground truth entries that are closer to the image border than this threshold will be discarded and not used in computation of the confusion matrix.

image rows and columns – The jtri files do not include information about the image size. In order to implement the border exclusion feature, the program must request this information from the user.

Outputs:

The output is a popup box that shows the confusion matrix entries. The user can save these to file by clicking on the *Save* button. Currently only an ascii save format is supported.

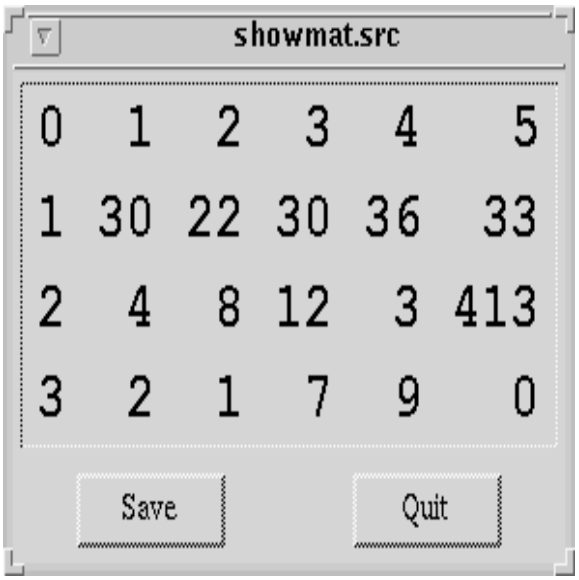


Figure 9.3: Confusion results.

9.6 Overlay

The *Overlay Tool* allows the user to display in successive frames a list of images overlaid with the corresponding jtri file. For the lists, the user may simply type the filenames separated by spaces into the entry box. Pathnames are required unless the images are in the current working directory. The user may also specify a list file using @-notation. Lists can contain a combination of @-notation and explicit filenames. The red builder button next to the entry box can be used to construct or select a list using one of several available methods. The jtri list may be omitted.

New frames are created for each image. Results are displayed in the new frames. Be careful when using this feature not to create too many frames because each frame consumes memory.

9.7 Chip Mosaic

The *Chip Mosaic* tool enables the user to extract chips (regions) from a list of images and display these in a mosaic. The chips can be sorted and annotated based on attributes in the jtri file such as radius, label, and value.

Inputs:

image list – This parameter specifies the set of images (in VIEW format) from which the chips will be extracted. The user may simply type the filenames separated by spaces into the entry box. Pathnames are required unless the images are in the current working directory. The user may also specify a list file using @-notation. Lists can contain a combination of @-notation and explicit filenames. The red builder button next to the entry box can be used to construct or specify a list using one of several available methods.

jtri list – This parameter specifies the set of jtri format marker files that indicate where to extract the chips. It is criticial that these files exactly correspond with the set of images specified in **image list**.

chip size – This entry specifies the size of the chips to be extracted.

chips per row – This optional entry specifies how many chips to place on one row in the mosaic. If omitted, the program tries to make an approximately square mosaic.

expression – This optional entry allows the user to associate a value with each of the chips in the mosaic. The value consists of an expression in terms of attributes defined within the jtri file such as radius, label, or value. The chips in the mosaic can be sorted according to the associated value through the **arrange** flag. The red builder button will display the available jtri attrbiutes and operators in a calculator format.

Note: If no expression is specified, the value 1 will be associated with every chip. To turn off the displayed text, use the *Toggle Marker Text* option under the *Labeling* menu button.

arrange – This option allows the user to rearrange the chip display based on the associated values. *Natural* means use the order of the chips as they appear in the jtri list, i.e., no sorting is applied.

Outputs:

The output is a mosaic image that will be displayed in the current frame (along with associated text values). Use the *Toggle Marker Text* option under the *Labeling* menu button to turn off the displayed text if desired.

9.8 Thumbnail

Given a list of images, this function creates an image of thumbnails in which each image is shown at reduced size. The *Thumbnail* tool is useful for browsing a large number of images at low resolution.

Inputs:

image list – This parameter specifies the set of images (in VIEW format) from which the thumbnails will be created. The user may simply type the filenames separated by spaces into the entry box. Pathnames are required unless the images are in the current working directory. The user may also specify a list file using @-notation. Lists can contain a combination of @-notation and explicit filenames. The red builder button next to the entry box can be used to construct or specify a list using one of several available methods.

thumbnail size – This entry specifies the size of each thumbnail in absolute pixel units. For example, 64 64 means that each image should be resized to 64 pixels by 64 pixels.

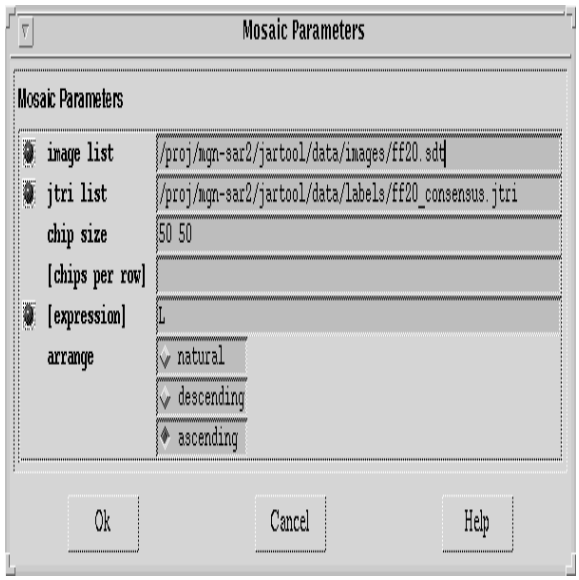
spoil size – This entry specifies the size of each thumbnail relative to the original size of the corresponding image. For example, 8 8 means that each original image should be reduced by a factor of 8 in each dimension. Exactly one of the inputs **thumbnail size** or **spoil size** must be provided by the user.

images per row – This optional entry specifies how many thumbnails to place on one row in the mosaic. If omitted, the program tries to make an approximately square mosaic.

Outputs:

The output is a mosaic image that will be displayed in the current frame.

(a)



(b)

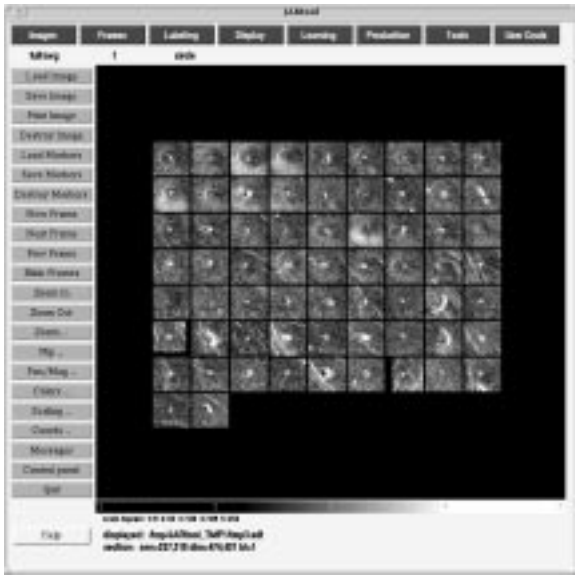


Figure 9.4: (a) Sample parameter values for the chip mosaic tool. (b) Result of running chip mosaic with the given parameters.

Chapter 10

User Commands

The *User Cmds* menu contains some simple image processing functions. The user can extend the functionality of JARtool by adding their own routines to this menu.

10.1 Image Min and Max

This function computes the minimum and maximum data values over the entire image. See also *Marker Analysis Tasks* in Chapter 2.

10.2 Image Mean and Standard Deviation

This function computes the mean and standard deviation of the data values over the entire image. See also *Marker Analysis Tasks* in Chapter 2.

10.3 Image Histogram

This function computes a histogram of the data values from the entire image. The resulting graph is displayed with a utility called *xgraph*.

10.4 Adding New Commands

Users can easily add their own commands to the *User Cmds* menu. The procedure works as follows:

1. Edit the analysis.cmds file which is located in $jartool/saotng/cmds/.

2. Copy and paste the code for one of commands listed above, e.g. *Image Histogram*.
3. Change the first line of code from “Image Histogram” to the name for your new routine as you want it to appear in the menu.
4. Change the fourth line to contain the name (with path) of your “callback”. The callback is the executable program or script that will be invoked when the user selects your menu option.
5. Your program or script can operate on the image and/or markers displayed in the current frame and can send results back to the display. Refer to the hypertext SAOtng documentation for complete details.

Part II

Sample Session

Chapter 11

Sample Session

We describe here a sample session with the JARtool system. You may want to try out these examples to get a feel for what you can do with JARtool and how to do it.

11.1 Loading an image

Click on the load image accelerator button. This will bring up a JARtool dialog box with the default image name:

$\$\mathrm{jartool}/\mathrm{data}/\mathrm{images}/\mathrm{ff20.sdt}$

Here, and throughout the sample session, $\$\mathrm{jartool}$ will be a placeholder for the directory in which the JARtool system is installed, typically something like

$/\mathrm{usr}/\mathrm{local}/\mathrm{JARtool}/$

Click the *OK* button of the dialog box and the image will be loaded.

Notice that the text information area at the bottom of the JARtool interface now contains the name of the displayed image as well as the size (dim =1024,1024) and blocking factor (bl=1). Two parameters in the JARtool config file specify the largest image size (default: 1280 $\times$ 1280) that can be loaded without blocking. Larger images will be block-averaged or down-sampled to fit within this size.

In the upper right corner of the image is a large blue square which shows a magnified view of the pixels under the cursor position. Go to the *Pan/Mag* accelerator menu button. Drag down and release on "Display Panner". The upper right corner now shows a reduced size version of your entire image. The cyan box shows the area currently appearing in the display. To change the area in the display, click inside the cyan box with the left mouse button and drag the box to the desired area

of the image. Release the left mouse button and the display will shift to show this section of the image.

You can see the entire image within the display area by clicking on the *Zoom out* accelerator button.

11.2 Labeling

Try out the labeling capability. The current marker shape appears under the *Labeling* menu button. The initial default is a circle. Change the marker shape to a rectangle by clicking on the *Labeling* menu button and dragging to "Set Default Marker Shape". This action will bring up a submenu called "Select Marker Shape". Drag over and select "Rectangle" by releasing the mouse button.

Now try to draw rectangles around all the volcanoes that you can find in the ff20 image. To draw a rectangle, click with the left mouse button to set one corner, drag, and release to set the other corner. For each of the rectangles you draw, indicate your confidence that the object is actually a volcano by clicking the right mouse button on the edge of the rectangle and pulling down to the "Set Type" option. This action will bring up a submenu titled "Set this Marker's Type (and color)". Choose the appropriate confidence level and release. Continue until you have labeled all the objects you think are volcanoes.

11.3 New Frame

Generate a new frame by clicking on the New Frame accelerator button. Select load image again and load the default ff20 image into this frame.

11.4 Load Markers

Click on the load markers accelerator button. This will bring up a JARtool dialog box with the default jtri marker filename $jartool/data/labels/ff20_consensus.jtri. Click on the *OK* button. *Zoom out* so that you can see the whole image in the display. The circles show all the volcanoes labeled by a pair of scientists. The colors of the circles indicate the scientists' confidence, while the sizes correspond to their estimates of the volcano diameter.

To visually compare the scientists' labeling with yours, click on the *Blink Frames* accelerator button. This will toggle the display back and forth between your labels

and the scientists' labels. Click on the *Blink Frames* button again to stop the blinking.

11.5 Save Markers

Go to frame 1 (your labels) using either the *Prev Frame* accelerator button or with the "Goto frame 1" option under the *Frames* menu bar button. Note that you may have stopped the blinking on frame 1, in which case you don't need to do anything. The current frame number is displayed below the *Frames* menu bar button.

After reaching frame 1, save your markers by clicking on the *Save Markers* accelerator button. The default filename that comes up is inappropriate so you will need to select a new filename. Click on the red builder button to bring up the file browser/file selection box (FSB). Erase the text in current selection (FSB) by typing **CTRL-a CTRL-k**. Now start typing the path to your home directory. Unfortunately, the standard UNIX $\sim$ convention is not supported here. As you type in the path, the listboxes below will display the contents of directories. You can single-click on any of these to navigate through the directory structure.

Eventually you will get to your home directory. Go to the *Hot Directories* menu button within the file browser and select "Add Current Directory". This will enable you to quickly go to this directory again in the future. Notice that your directory is now added as an option under the *Hot Directories* menu. Finish typing the filename you want for your file, say $\mathrm{ff20\_mylabels.jtri}$. When you are done, click on the *OK* button.

11.6 Learning

Now let's try out the JARtool learning function. Click on the *Learning* menu button and release on the "LEARNING" menu option. This action will bring up a parameter input GUI (PIG) titled "Learning Parameters".

Click on the red builder button next to **image list** and select *XlistEditor*. Under the *File* menu button, select "Load File List". This action will bring up a JARtool dialog box. Click the *OK* button to accept the default entry. The files from this list will will now appear in the *XListEditor* under "Current List". Adjust the scroll bar beneath "Current List" so that you can see the filenames. Double-click on $\mathrm{ff20.sdt}$ to remove this item from the current list.

Again go under the *File* menu button, but this time select "Save File List". Use the red builder button to invoke the file browser. Under *Hot Directories* select your home diredctory. Type the filename mylist in the "Current selection" entry box

and click *OK*. This will save your current list to file and bring you back to the *XListEditor*. Click *OK* here as well.

Notice that the **image list** parameter in the PIG is now set to the list you just saved. The @ symbol tells the learning programs that mylist is not itself the name of an image, but instead is the name of a file which contains image names. We refer to this method of specifying a list as @-notation.

We also need to modify the **jtri list**. Click on the red builder botton and select *XListEditor*. Follow the same procedure as with **image list**, i.e., under *File* load the default file. Double-click on ff20_consensus.jtri to remove this entry from the "Current List" window. Note that if you accidentally remove ff21_consensus.jtri, you can just double-click on this entry in the third window to add it back to the current list.

After you have properly deleted ff20_consensus.jtri, click on the *OK* button. Since we didn't bother to save this list to a file, all the list entries appear in the **jtri list** entry box. Most of the entries scroll off the end, but you can drag with the middle mouse button to see that they are really there.

The final thing to do before starting the learning process is to change the **output directory**. Click on the red builder button and select file browser. Use the *Hot Directories* menu button to go to your home directory. Type the name of a directory to use for the output of learning, say learning_test. Click the *OK* button of the file browser.

Now you are ready to run learning. Click on the *OK* button of the "Learning Parameters" PIG. You should immediately see a busy box which contains a blue jar filling with water. The busy box indicates that the learning process is running. It may take a while (1-10 minutes) to run learning depending upon the processing speed and memory of your machine. If you are interested in your progress, the window from which you started JARtool will show some checkpoint messages from the learning procedure. When learning is finished, the jar of dripping water will disappear.

11.7 Production

Click on the *Production* menu bar button and select the "PRODUCTION" option. This action will bring up the "Production Parameters" PIG. You will need to change some of the parameters. To do this, you can use the builder buttons or type directly into the entry boxes.

- Change the image list entry to $jartool/data/images/ff20.sdt where as usual $jartool should be replaced by the path to the toplevel directory

where JARtool is installed. Note that there should not be an @ sign in front because this is itself the name of an image.

- Change "learning directory" to $\$\mathrm{your\_home/learning\_test}$ where $\$\mathrm{your\_home}$ is the path to your home directory.
- Change the output directory to $\$\mathrm{your\_home/production\_test}$.
- Make sure that the **Auto-Display** control flag is turned on. (If the square button next to **Auto-Display** is raised, click to turn the flag on.)
- Also, make sure that the **Auto-Insert** control flag is turned on.

After all the parameters are set properly, click on the *OK* button of the "Production Parameters" PIG. You should immediately see a jar filling with water to indicate that production is running.

When production is completed the $\mathrm{ff20}$ image will automatically be displayed in the next available frame (frame 3 if you have been following along through the example). Use the *Zoom out* accelerator button to see the entire image. There should be be a set of red and blue squares overlaid on the image. All squares correspond to areas that the focus of attention (FOA) algorithm identified as interesting. The blue squares, however, were rejected by the classifer as non-volcanoes while the red squares were declared to be volcanoes. There is a threshold in the production user parameter file that can be adjusted to make the classifier more aggressive or more conservative.

11.8 JTRI Filter

To see only the red squares, go to the *Tools* menu and select the "JTRI Filter" option. The default entry for **jtri list** is $\mathrm{sao:}$, which tells the system to get the currently displayed markers. The default output is also $\mathrm{sao:}$, which tells the system to put the filtered markers back on the display. The default filter expression is "(L == 1)", which selects only those markers that have a label equal to 1. Since the red squares have label 1, we can just use the default settings. Click on the *OK* button. The marker overlay will be redrawn with only the red squares.

Save the JARtool markers to a file using the *Save Markers* accelerator button. Save the JARtool markers as $\$\mathrm{your\_home/JAR\_markers.jtri}$.

11.9 Chip Mosaic

Under the *Tools* menu, select "Chip Mosaic". This action will bring up a "Mosaic Parameters" PIG. Click on the red builder button next to **expression** to bring up a

calculator-style menu of jtri attributes and operators. (You may need to move the attributes and operators menus around slightly on your workspace to see both.) Click on "LABEL" in the attributes menu. The letter "L" will appear in the expression entry box. Go back to the PIG and click the "ascending" option (one of the buttons next to the **arrange** parameter). Next click the *OK* button of the PIG.

The tool will generate a collage of chips which were extracted from locations identified by the scientists as potential volcanoes. The subjective confidence labels assigned by the scientists are shown in cyan. Notice that we have displayed the chips in ascending order by label. If the contrast of the displayed image is not so good, you can modify it by moving the cursor into the display area and dragging with the right mouse button until the contrast is satisfactory. You can toggle the display of the text by selecting the "Toggle Marker Text" option under the *Labeling* menu button.

11.10 Scoring

Let's see how well the JARtool system labels compare with those of the scientists. First we will just blink the display between frames 2 and 3. The *Blink Frames* accelerator button will cycle through frames 1, 2, and 3 so we will need to use the *Control Panel* accelerator button to set up a different blink protocol.

On the *Control Panel*, there is an area labeled "Blink" with frames set to 1,2,3. Clicking on each of the small grey boxes will increment through the frame numbers (similar to setting a digital alarm clock). Change the frames to 2,3 and then click the "Blink Frames" box. You can change the blink rate using the arrow buttons.

An alternative way to visualize the results is to load the scientists' markers into the same frame as the JARtool system markers. Go to frame 3 and click the *Load Markers* accelerator button. The default filename should be correct, so just click *OK*. The scientists' markers appear as circles, while the JARtool system markers appear as squares. Objects marked with both a circle and a square were identified as volcanoes by both the scientists (subject to the confidence statement) and the system. Circles with squares are "misses" by the system and squares without circles are "false alarms".

We can quantify how well the system works by computing a confusion matrix between the system and the scientists' consensus labeling. Under the *Tools* menu, select "Confusion". This action will bring up the "Confusion Parameters" PIG. You will need to make the following changes:

- Change the first jtri list parameter (detections) to $\$\text{your\_home}/\text{JAR\_markers.jtri}$ (no @ sign).

- Change the label limits to "1 1".
- Change the second jtri list (ground truth) to $\$\mathrm{jar\_home/data/labels/ff20\_consensus.jtri}$ (again no @ sign).
- The label limits in this case should be "1 4".

After making these changes, click on the *OK* button. In a few seconds, a gray window will popup showing the confusion matrix. The (i,j) entry is interpreted as the number of objects labeled i by the system and labeled j by the consensus. The (2,j) entries represent objects not identified by the system but identified by the consensus of scientists ("misses"). The (1,5) entry corresponds to objects identified by the system but not by the scientists ("false alarms").

Keep in mind that for the Magellan data, the "ground truth" labels are subjective. In other words, different scientists will not identify exactly the same set of objects as volcanoes. Hence, in evaluating the system performance we must compare how well the system agrees with "ground truth" to how well an individual scientist agrees with "ground truth".

As an exercise, calculate the confusion matrix between your labels and the ground truth. How did you do compared to the JARtool system?

11.11 Database

Getting Started

To start the query tool, select *Query Tool* from the *Database* menu on JARtool. (The database must already be up and running. Refer to Appendix A for instructions on setting up the database.) Next connect to the Sample database by choosing the appropriate option from the *Databases* menu on the *Query Tool*.

Listing Images

From the *Views* menu on the *Query Tool*, activate the *List Records* submenu. From this submenu select "ImageFiles". This should list all the records in the images table (there are only four). Each record should contain the name of the image, the latitude and longitude range, and the look direction.

If you would like to view all of the listed images, select "Display Images" from the *Images* menu on the *Query Tool*. This will load the four images into four different frames.

One can also save the results as an image list file. Select the "Save Results (as Image List)" option from the *Images* menu on the *Query Tool*. Enter the save filename and click the *Ok* button.

Besides listing all the images, one can also list images which satisfy a particular criterion. Click on the *Img(Lat,Lon)* accelerator button. This will display an SQL query in the Query Window.

Edit the text in the Query Window. You will need to substitute actual values for the <LAT> and <LON> placeholders.

Original:

```
#define LAT <LAT>
#define LON <LON>
```

After Substitution:

```
#define LAT 31
#define LON 332
```

Now click the *Execute* button. This will list all the images which contain the specified point on the planet. In this case, there is only one image in the sample database that contains this point. To view this image in JARtool, notice that the filename is displayed as a hyperlink (blue text). Simply click on the hyperlink.

Listing Labeling Sessions

To find out which images have been labeled, which person (or algorithm) did the labeling, and the jtri file produced, activate the "List Records" submenu from the *Views* menu. Then select "LabeledImages".

From the *Results* menu, select "Display Images and Regions". This will display the four images and corresponding jtri files in four different frames. If you used the **Auto-Insert** option during the production part of the sample session, then there will be a fifth entry pairing the ff20 image with the jtri file generated by production.

One can also list only the labeling sessions that satisfy a particular criterion. Click the *ImgRgn(Auth)* accelerator button. An SQL query will appear in the Query Window. Substitute

```
#define AUTHORNAME <AUTHORNAME>
```

with

```
#define AUTHORNAME "('saleem')"
```

and click the *Execute* button. All the files that were labeled by saleem should be listed along with the names of the region files he produced.

Similarly to the image hyperlinks, if you click on the name of a region file, the region file will be loaded into JARtool.

Listing Labels

If you list all the records in the *LabelsWCS* view, it will list all the labels in WCS coords; the *LabelsPIX* view lists all the labels in pixels.

First load ff05.sdt into the JARtool display area. (One way to do this is to list all the image files, and click on ff05.sdt). Click *LabPx(Img, Auth)*. An SQL query will be displayed, which you must edit. Substitute

#define IMAGENAME <IMAGENAME>
#define AUTHORNAME <AUTHORNAME>

with

#define IMAGENAME '$jartool/data/images/ff05.sdt'
#define AUTHORNAME 'saleem'

Remember to replace $jartool with the proper path to the JARtool root directory.

Click *Execute* and all the labels drawn by saleem on ff05.sdt will be listed. Click *Display Regions* and the labels will appear on the JARtool display.

The color is dependent on the value of the *LabelName* field. This value can be changed under the *Regions* menu button. For example, to compare the labels of two authors, you might display one in red and the other in green. Also all fields which do not describe the shape, position or size of the marker will be attached as text. The attached text can be hidden by selecting "Toggle Marker Text" from the *Labeling* menu in JARtool.

The listed regions can also be saved to a JTRI file. From the *Regions* menu on the *Query Tool* select "Save Results (as JTRI file)", enter the desired output filename and click *Ok*.

Listing Objects

From the *Views* menu, list records in *Objects*. This will list the position of all volcanos in the database. To do a more interesting query, click *Obj(Img)*. Edit the query by substituting

#define IMAGENAME <IMAGENAME>

with

#define IMAGENAME '$jartool/data/images/ff05.sdt'

Click *Execute* query to list all the volcanos in the database contained in this image.

Now click *Display Regions* and the listed volcanos will be drawn as points.

Note that the *Obj(Img)* and other *Obj(·)* queries are not applying any of the detection/recognition algorithms to the image. They are only conulting the database to see if someone (or some algorithm) has recorded labels on the designated image.

Also, the objects may not actually correspond to true volcanoes, e.g., an algorithm may have generated false alarms which were then inserted into the database.

Acknowledgements

The authors wish to thank the developers of SAOtng at the Smithsonian Astophysics Observatory for their responsiveness to our requests for improvements and new features. We also thank Jayne Aubele and Larry Crumpler, our planetary geologist collaborators. Jayne and Larry evaluated early versoins of the system and provided valuable feedback that helped guide the development of the version 2.0 system.

In addition, we wish to acknowledge the following people who were involved either in earlier versions of the JARtool interface or in algorithm development and testing: Lars Asker, Kevin Cherkauer, Victoria Gor, Steve Hyland, John Loch, Mike Turmon, and Jennifer Yu.

Research and development of the system described in this manual was carried out in part by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

Bibliography

[1] M.C. Burl, L. Asker, P. Smyth, U.M. Fayyad, P. Perona, J. Aubele, and L. Crumpler, "Learning to Recognize Volcanoes on Venus", *(in review)*, (September 1997)

[2] M.C. Burl, U.M. Fayyad, P. Perona, and P. Smyth, "Trainable Cataloging for Digital Image Libraries with Applications to Volcano Detection", *CNS Tech Report: CNS-TR-96-01*, Calif. Inst. of Technology, October 1996.

[3] U.M. Fayyad, P. Smyth, M.C. Burl, and P. Perona, Chapter: "Learning to Catalog Science Images", In *Early Visual Learning*, Oxford University Press, pp. 237-268, 1996.

[4] M.C. Burl, U.M. Fayyad, P. Perona, P. Smyth, and M.P. Burl, "Automating the Hunt for Volcanoes on Venus", In *IEEE Comp. Society Conf. on Computer Vision and Pattern Recognition*, pp. 302-309, Seattle, WA, June 1994.

Part III

Technical Appendices

Appendix A

JARtool Installation

In order to install JARtool, first make sure that you have the appropriate Tcl/Tk code (wish4.2 and tclsh7.6 or newer) installed on your system. Also, be sure that the directory containing the wish executable is in your path. Next download the following files:

1. saord.tar.gz – the SAOtng source code.
2. postgres.tar.gz – the Postgres source code.
3. jartool.tar.gz – the JARtool source code.

and follow these steps:

1. Install SAOtng on your system as follows:
   (a) Create a directory where you would like to install SAOtng. (From here on this directory will be refered to as SAOtngHome).
   > mkdir $SAOtngHome
   (b) Move the saord.tar.gz file to $SAOtngHome.
   > mv saord.tar.gz $SAOtngHome
   (c) Change directory to $SAOtngHome.
   > pushd $SAOtngHome
   (d) Uncompress the saord.tar.gz file.
   > gunzip saord.tar.gz
   This command will produce a saord.tar file.
   (e) Unpack the saord.tar file.
   > tar xvf saord.tar
   This will produce a saord subdirectory.
   (f) Change directory to saord/Doc and follow the instructions in the BUILD file. (Note that you do not need to install the IRAF tools. Also you should not do make strip, keep the source code around).
   (g) After installing SAOtng, you must put $SAOtngHome/saord/bin in your path.

(h) Change directory back to the directory where you downloaded the tar files.
> popd

2. Next you must install Postgres on your system. To do this

(a) Create a directory where you would like to install Postgres. (From here on this will be refered to as PostgresHome).
> mkdir $PostgresHome
(b) Move the postgres.tar.gz file to $PostgresHome.
> mv postgres.tar.gz $PostgresHome
(c) Change directory to $PostgresHome.
> pushd $PostgresHome
(d) Uncompress the postgres.tar.gz file.
> gunzip postgres.tar.gz
This command will produce a postgres.tar file.
(e) Unpack the postgres.tar file.
> tar xvf postgres.tar
This will produce a postgresql-v6.1.1 subdirectory.
(f) Change directory to postgresql-v6.1.1 and follow the instructions in the INSTALL file. (You must create the directory to hold your data, initialize the database and run the postmaster application on some machine which we will refer to as HOST).
(g) After installing Postgres, you must put $PostgresHome/bin in your path.
(h) Change directory back to the directory where you downloaded the tar files.
> popd

3. Now you can install JARtool on your system as follows:

(a) Create a directory where you would like to install JARtool. (From here on this will be refered to as JARtoolHome).
> mkdir $JARtoolHome
(b) Move the jartool.tar.gz file to $JARtoolHome.
> mv jartool.tar.gz $JARtoolHome
(c) Change directory to $JARtoolHome.
> pushd $JARtoolHome
(d) Uncompress the jartool.tar.gz file.
> gunzip jartool.tar.gz
This command will produce a jartool.tar file.
(e) Unpack the jartool.tar file.
> tar xvf jartool.tar
This will produce a jartool subdirectory.
(f) Change directory to the jartool subdirectory.
(g) Now compile all the source code.
> make PSQL=$PostgresHome SAOTNG=$SAOtngHome HOSTNAME=$HOST
(h) Install all the executables.
> make install PSQL=$PostgresHome SAOTNG=$SAOtngHome HOSTNAME=$HOST
(i) Clean up.
> make clean
(j) Put the $JARtoolHome directory in your path.
(k) Change directory back to the directory where you downloaded the tar files.
> popd

Appendix B

VIEW Format

Images used by the JARtool learning and production components are required to be stored on local disk in what is known as VIEW format. This format consists of two files per image. The ASCII **.spr** file contains header-type information (number of dimensions, number of rows, number of columns, data type, etc.), while the binary **.sdt** file contains the pixel data. When supplying image names to any of the various JARtool routines, specify the filename with the .sdt extension.

Here is a sample **.spr** file:

```
2
1024
0.000000
1.000000
1024
0.000000
1.000000
0
0.000000
0.000000
```

This file indicates that the data is 2-dimensional (1024 X 1024) and in byte format (0). The real-valued numbers originally were used to designate the origin and sample spacing, but these are no longer supported.

Appendix C

JTRI Format

Labeling information provided by scientists or generated by automatic algorithms is represented in JARtool using *jtri format*. This format evolved from the native region format used in earlier versions of SAOimage. The jtri format contains one row for each detected or labeled object. A typical entry might be as follows:

$\mathrm{CIRCLE}(352,\ 438,\ 16.639999)\ \#2\ \text{-}1\ \$2$

This entry specifies a circle centered at $(352, 438)$ with radius 16.64 pixels. The #2 and $2 indicate the color and label (scientist's confidence) of this object. The label may also be followed by another field indicating time (or other attribute) as well as a text comment (preceded by #). Multi-line comments are possible by using backslash-n to represent line breaks.

Appendix D

Learning Parameters

To change the behavior of the learning algorithm, you will need to copy the default learning_user.pars file from $jartool/analysis/defaults/
and edit the parameters with a text editor such as emacs. Cross-validation and Ruler are no longer supported so be sure to keep xval = 0 and ruler = "no".
We recommend experimenting only with the following parameter values:

fullresize – The size of the images at full-resolution. This parameter is used to scale the coordinates of objects detected on reduced-size versions of the image back to the full-size coordinates.

foa_threshold – This threshold determines how aggressively the focus of attention will be in declaring things to be candidate volcanoes. Lowering the threshold will make the FOA more aggressive; raising the threshold will make the FOA more conservative. If the threshold is too low, too many things will be detected, while if the threshold is too high, no objects will be detected, so it may be necessary to experiment to find an appropriate value. A good rule of thumb is that the FOA threshold should be adjusted so that 80–95% of the true objects are detected.

patchsize – This parameter specifies how large a patch around each object to extract and give to the learning algorithm. The patch must be large enough to contain a distinctive pattern.

spoilfactor – As a computational speed-up, much of the processing is done on reduced resolution (block-averaged) images. This parameter specifies the block size.

spoilpatchsize – This parameter is just the ratio of the **patchsize** and **spoilfactor**. The program should be able to compute this value, but due to a bug you must do the calculation yourself and enter the proper value.

llbl, ulbl – These parameters specify the lowest and highest labels in the ground truth jtri files (default = 1 4).

gauss_prob_thresh – This threshold specifies how aggressive the classifier is in declaring candidate regions to be volcanoes. At low thresholds, the classifier is very aggressive; at high thresholds, the classifier is conservative.

border_thr – This parameter excludes a border around the images. The filtering and feature measurement operations use templates that may extend off the edge of an image. The resulting values are often abnormal and should be excluded. The parameter value should be set to half the **patchsize**.

cluster_thr – Nearby detections are grouped and attributed to the same object. This threshold specifies how close two points must be to be considered "nearby".

scoring_thr – This parameter specifies how close a detection must be to a ground truth location to be counted as correct (used to identify not-volcano examples for classification training).

Appendix E

Production Parameters

To change the behavior of the production algorithm, you will need to copy the default $\mathrm{production\_user.pars}$ file from $\$\mathrm{jartool}/\mathrm{analysis}/\mathrm{defaults}/$
and edit the parameters with a text editor such as emacs. Cross-validation and ruler are no longer supported so be sure to keep $\mathrm{xval} = 0$ and $\mathrm{ruler} = \text{``no''}$.
Basically, all the parameter values should be the same as you used during the learning phase. The only parameters that you might want to be different are **foa_threshold** and **gauss_prob_thresh**.

Appendix F

Known Bugs in JARtool

Busy Box Kill Button: There is a potential bug with the busy box kill button. When clicked the box does a ps -l system command to list currently running processes and then it tracks down all children of the primary process which is to be killed. However, in the time between performing the ps command and the time when the child processes are actually killed, it is possible that new processes could be spawned; these will not be killed properly.

Hot Directories: Hot directories added by different users may be placed in the same central file. There is no easy mechanism for deleting a hot directory except for editing the file.

UNIX tilde notation: The standard convention that a tilde ($\sim$) represents a user's home directory and a tilde in front of a username represents that user's home directory is not supported in JARtool.

Menu Bar Submenus Disappear: The menu bar submenus disappear if there is no image in the current display area, e.g., if an image load fails or if the *Destroy Image* accelerator button is used. Loading an image will cause the submenus to reappear.

Hypertext Help: From the middle of one help page, if you follow a hyperlink to another help page and then hit **Prev** to go back, you are returned to the top of the first page rather than your previous point midway through the document.

FOA and Classifier Scores Not Included in JTRI Files: The jtri file format does not easily permit the inclusion of extra information such as the scores produced by

the focus of attention (FOA) or classification algorithms. Hence, the only way to see the results at different threshold settings is to modify the $\mathrm{production\_user.pars}$ file and rerun production. If the scores were included in the jtri files, then one would have been able to run the system at low threshold settings and use the *JTRI Filter* tool to interactively determine good thresholds. Another consequence of this bug is that the **VALUE** attribute is not supported in *JTRI filter* or *Chip Mosaic*.

Query Tool Resizing and Scrolling: Resizing the *Query Tool* to incease the amount of text displayed in the Results Window does not work properly due to a bug in the Tk text widget. There may also be slight errors in the text scrolling (e.g., only half the bottom line may be visible) due to the same bug.

Database Deletions: Users must be extremely careful when deleting records (rows) from any of the database tables because these operations may affect other tables. *Postgres* does not support triggers, so the effects upon other tables cannot be propagated appropriately. Consider, for example, deleting a row from the *Hometown* table discussed in Chapter 6. The $\mathrm{Hometown\_ID\_number}$ for the deleted row may still appear in other tables such as the *Employees* table. These other tables will now contain stale and/or corrupted data.