ARL-SR-0384 ● Oct 2017

US Army Research Laboratory

Symbolic and Sub-Symbolic Robotic Intelligence Control System (SS-RICS) User's Manual

by Troy Kelley, Eric Avery, and Sean McGhee

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-SR-0384 ● OCT 2017

US Army Research Laboratory

Symbolic and Sub-Symbolic Robotic Intelligence Control System (SS-RICS) User's Manual

by Troy Kelley, Eric Avery, and Sean McGhee
Human Research and Engineering Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
October 2017	Special Report	13 January 2011–13 January 2016

4. TITLE AND SUBTITLE
Symbolic and Sub-Symbolic Robotic Intelligence Control System (SS-RICS) User's Manual

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Troy Kelley, Eric Avery, and Sean McGhee

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-HIA-IB
Aberdeen Proving Ground, MD 21005-5425

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-SR-0384

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This is a user's manual for the Symbolic and SubSymbolic Robotics Intelligence Control System. The manual explains the overall use and operation of the robotics architecture, the different setups and connections, the simulator and real-world operation, and the code-development environment for creating programs for the robot. A wide variety of topics is covered, from code deployment to the theoretical issues involving symbolic and sub-symbolic knowledge representation. It is intended to be a general platform for robotics development with an emphasis on cognitive mechanisms for robotics control.

15. SUBJECT TERMS
robotics, user's manual, SS-RICS, cognition, cognitive architectures

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Troy Kelley
Unclassified	Unclassified	Unclassified	UU	248	19b. TELEPHONE NUMBER (Include area code) 410-278-5869

Contents

List of Figures

List of Tables

1. System Overview

The artificial intelligence (AI) community has struggled for 30 years with the question of how to reproduce the thought patterns of the human mind. This quest has resulted in 2 distinct paradigms. The first camp uses symbolic representations to drive intelligent systems, and the second focuses on a mathematical approach using distributed representations constructed with structures such as neural networks. During the same time, the psychological community has used cognitive architectures in an attempt to reproduce the thought patterns of the human mind. Such cognitive architectures include the Adaptive Control of Thought Rational (ACT-R) (Anderson and Lebiere 1998) and Soar (Newell 1992). The primary goal of these systems is to model human mental processes, human knowledge, the nature of that knowledge, and how the knowledge is used and accessed. Both Soar and ACT-R have been designed on similar grounds, as symbolic production systems in an attempt to define a unified theory of cognition and merge a number of cognitive phenomena into a single architecture. The Symbolic and Sub-Symbolic Robotic Intelligence Control System (SS-RICS) is an attempt to unify symbolic and sub-symbolic mechanisms. In other words, merge large knowledge-acquisition systems with cognitive psychological principles while also utilizing distributed statistical intelligence techniques.

The challenge of merging these 2 approaches and understanding human cognition has fueled the design of SS-RICS. By applying cognitive psychological principles to robotics, we believe it is possible to produce a system that is capable of autonomous operation and high-level interaction with human operators. This system merges high-level symbolic representations of objects and activities into a production system processor driven by high-level goals as well as low-level parallel processing of input sensory devices. The objectives of SS-RICS are not just the integration of symbolic and sub-symbolic techniques, but also the utilization of additional techniques from AI, such as machine learning, neural networks, and semantic networks.

Conceptually, SS-RICS is organized into 3 layers. One layer serves as the Long-Term Memory (LTM) and, in practical terms, it is a relational database or a semantic network. As of this writing we are interested in using ConceptNet (Liu and Singh 2004) and have made some progress integrating ConceptNet into SS-RICS (see p. 11, for more details). ConceptNet allows SS-RICS to relate similar objects to each other (i.e., dogs and cats are similar) and allows SS-RICS to query the database for information about objects. For example, we can query ConceptNet and find that “Sweet” is a “PropertyOf” “Apple”.

The second layer within SS-RICS is the production system, which serves as the Working Memory (WM) component of the architecture. The production system matches current goals and rules to facts within WM (not LTM). We are currently working on processes that pull information from LTM into WM by using spreading activation (Anderson and Lebiere 1998). For example, if the robot defined a goal of going through a door, it would pull goals from LTM associated with going through a door, as well as other goals, for going down halls and maneuvering in rooms. This requires productions to have activation to be stored as memories, which is different from ACT-R, which does not assign activation values to productions. This work is still in progress.

The third layer is the sub-symbolic layer, which serves as the perceptual component of SS-RICS. The key aspect of the perceptual layer is that all of the sub-symbolic systems are running in parallel and can be turned off if they are using too much processing power. This makes SS-RICS relatively scalable in terms of real-time processing. The sub-symbolic layer provides information to the production system in a continuous fashion. As SS-RICS is developed, newer sub-symbolic systems can be added or removed as necessary. We have tried to make this component as modular as possible.

SS-RICS is currently designed to interact with MobileRobots' Pioneer platform of robots, the SRV-1 robot and the iRobot PackBot. SS-RICS uses MobileRobots' Advanced Robot Interface for Applications (ARIA) application program interface (API) to communicate with the robotic platform and simulators via either a serial or Transmission Control Protocol (TCP)/IP connection. The implementation of SS-RICS was designed and developed using both managed and unmanaged C++ using Microsoft Visual studio 2003, 2008, and 2010. SS-RICS is currently being developed under a joint effort between US Army research laboratories and Towson State University.

SS-RICS has a very simple graphical user interface (GUI) for goal development. We have tried to make the goal editor accessible to the novice. Some understanding of production systems would be helpful for developing goals within SS-RICS but it is not absolutely necessary. There is also a very powerful text editor for developing goals within SS-RICS. This is for the more advanced user. We have tried to keep this syntax as simple as possible as well. It is similar to production systems in ACT-R, with a left- and right-hand side or a consequence and antecedent. We have implemented this in Visual Studio to take advantage of Visual Studio's editing capabilities like syntax-coloring, cut-and-paste facilities, and real-time error reporting.

As of this writing there are some significant differences from ACT-R that the modeler should be aware of, as listed in the following. These will probably change as SS-RICS develops further.

- Productions have activations because we are interested in storing productions as memories, so they must have activation values.
- Production utility: Currently there is no production utility mechanism. We are looking at developing one and are using activation instead. Since productions have activations, we can use that as a utility function.
- Proceduralization: There is no proceduralization (where productions get collapsed together to form new productions). Eventually we would like productions to be collapsed into sub-symbolic mechanisms, but that is a work in progress.
- Spreading activation: There is no spreading activation because we are working with a very large LTM. This will be implemented eventually.
- Conflict resolution: In SS-RICS, conflict resolution is based on activation (the production with the highest activation is selected). We are looking at other solutions for conflict resolution including using Decision Field Theory (DFT) (Busemeyer and Johnson 2006).

2. Decision Field Theory

We have been interested in using DFT as an attention mechanism, and this work is continuing. DFT, currently accessible within SS-RICS, is a framework for applying decisions to an evidence accumulation model. It is defined as a continuous-time dynamic process whereby at every point in time, a stochastic attention mechanism focuses on one of the features/outcomes of the choice situation. All choice alternatives are compared against each other along this particular feature dimension, and a preference state for each alternative is incrementally adjusted. This process continues until a preference value crosses some threshold, at which point the alternative with the maximum preference is selected.

Figure 1 demonstrates how preferences change over time in a decision between 3 given options. The blue “rule” is finally selected in this example.

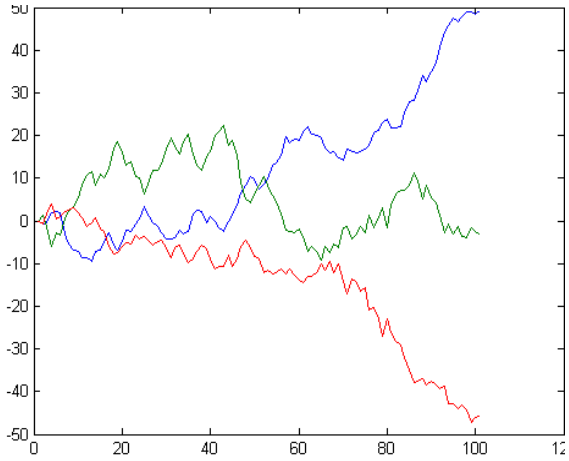


Fig. 1 DFT preference progress graph (bottom axis = time)

The topmost option is the first to hit the threshold, which ends the decision process. At that point an action is taken to indicate that the blue line is selected.

3. Getting Started

3.1 Installation

The SS-RICS installation is based on the Microsoft installer, therefore the install process should be straightforward for many Windows users.

3.1.1 Zip File

- Extract the files from the zip file to a directory on your computer.
- Open My Computer.
- Browse to the directory where you extracted the contents of the SS-RICS zip file.
- Find the setup.exe file.
- Double-click on the setup.exe file.
- Follow the onscreen setup instructions.

3.1.2 CD

- Open My Computer.
- Browse to the CD-ROM drive on your computer.
- Find the setup.exe file.

- Double-click on the setup.exe file.
- Follow the onscreen setup instructions.

3.2 Starting SS-RICS

Open SS-RICS by selecting SS-RICS Interface from the SS-RICS group in the Start Programs menu. This will open SS-RICS's main interface. Then a dialog will appear so that you may specify how you would like to connect to the robot (Fig. 2).

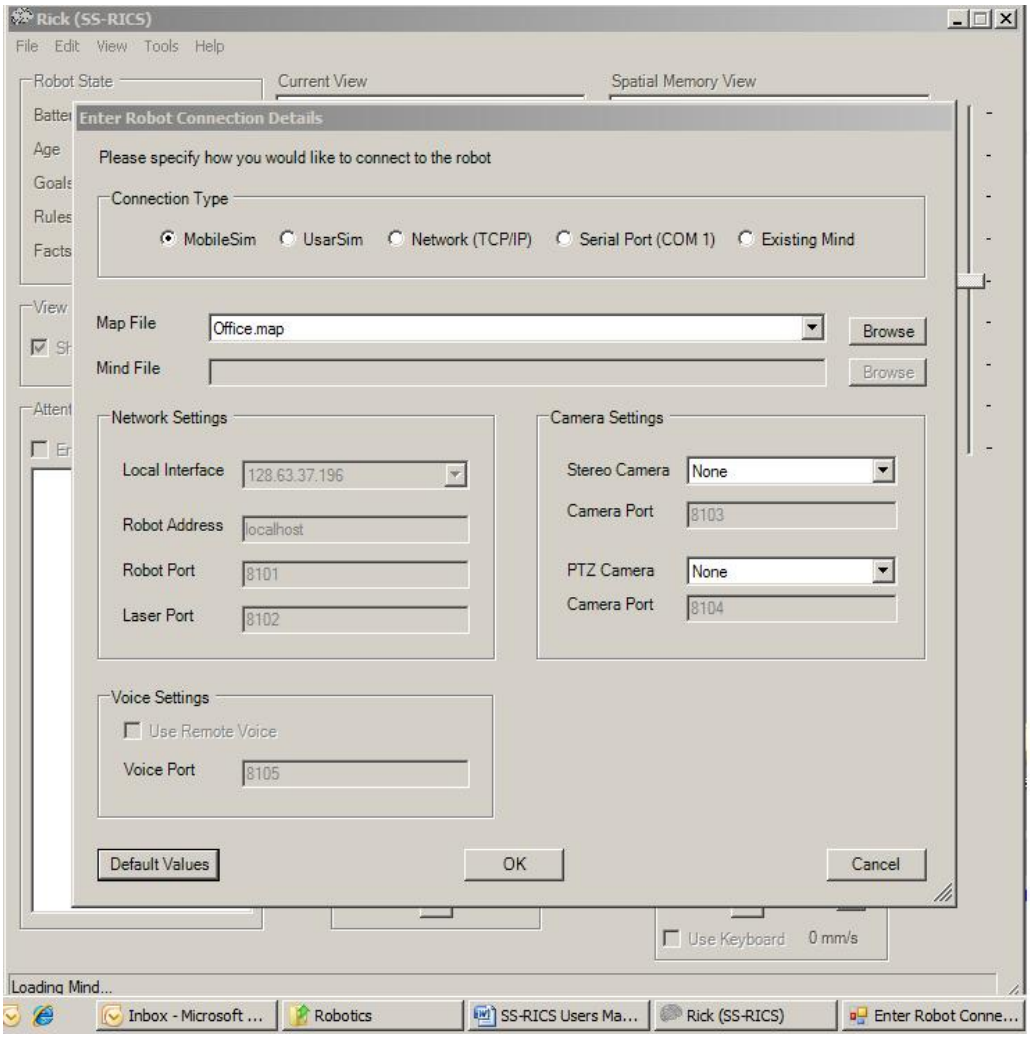


Fig. 2 SS-RICS connections/camera options

This interface will allow you to set up your connection type:

- MobileSim is the ARIA simulator.
- UsarSIM is a 3-D environment

- NetworkTCP/IP allows you to connect the robot to a server.
- Serial Port—COM1—allows for a serial port connection.

After you have selected a connection type, SS-RICS will attempt to connect to the robot or simulator. During this connection process the interface is disabled. If SS-RICS is unable to connect to the robot/simulator, a dialog will be displayed notifying you of the connection issue. If you are using an actual robot you may notice that the interface is disabled while the Sick Laser system warms up. This is normal; the interface will become active as soon as the laser is online.

4. User Interface

4.1 Mind Interface

4.1.1 Overview

SS-RICS's main interface allows you to manage the robot operation from a single screen and acts as a portal to the rest of the system. This window includes the ability to operate the robot and its onboard camera, visual output of sensory data collected by the sub-symbolic processors, and the current operating state of the robot/system. You can also execute goals by double-clicking on them in the attention window. The interface's main menu (Fig. 3) gives you access to other parts of the system, which allow for more detailed control of the system.

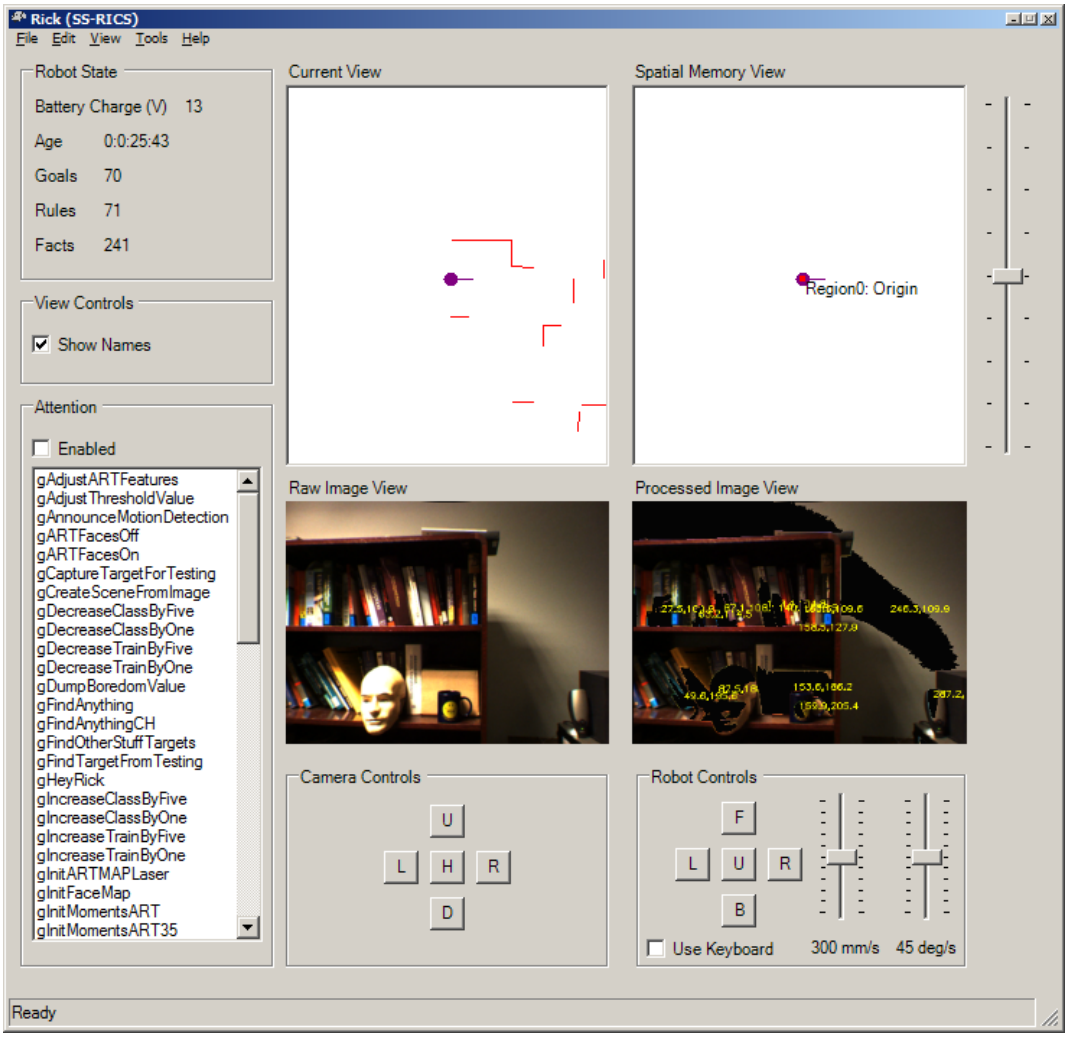


Fig. 3 SS-RICS main user interface

4.1.2 Robot State

The robot-state section of the interface displays status information available from the robot, such as battery voltage and robot location. The age (how long it has been active in the current session) of the mind and number of rules and facts in the mind are also displayed.

4.1.3 Current View

This display area is the current data captured from the SICK, Inc., laser guidance system and sonar devices when enabled. This area may also display other data derived from the SICK data such as corner and gap locations. Gap, corner, and parallel lines are displayed in different colors so you can distinguish them from the simple line data.

4.1.4 Spatial Memory View

This displays spatial memory data generated by the sub-symbolic processors. This area also displays gaps, corners, lines, and regions in different colors so you can distinguish them from the simple line data. You can also turn these off and on by using the Show Names toggle switch.

4.1.5 Attention Window

The attention process is still under development, but it is intended to be a visual representation of the active goals in the system as they are processed by a high-level "attention" process. In the interim, you can use this window to double-click on these goals to activate one.

4.1.6 Robot Controls

These buttons allow you to manually control the robot and will override the low-level robot functions used by the symbolic and sub-symbolic system components. Therefore, you should be able to correct the action of the robot even when it is being controlled by the system subcomponents.

Control Buttons

- F button: while pressed the robot is moved forward.
- B button: while pressed the robot is moved backward.
- L button: while pressed the robot is turned left.
- R button: while pressed the robot is turned right.
- U button: when pressed the robot turns around.
- Left slider: governs the speed of the robot.
- Right slider: governs the turning angle of the robot.

These controls can be detached from the main interface panel by right-clicking inside the box containing the controls (Fig. 4).

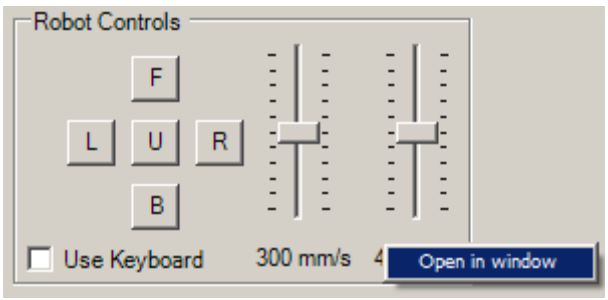


Fig. 4 Right-click menu for robot controls

4.1.7 Camera Controls

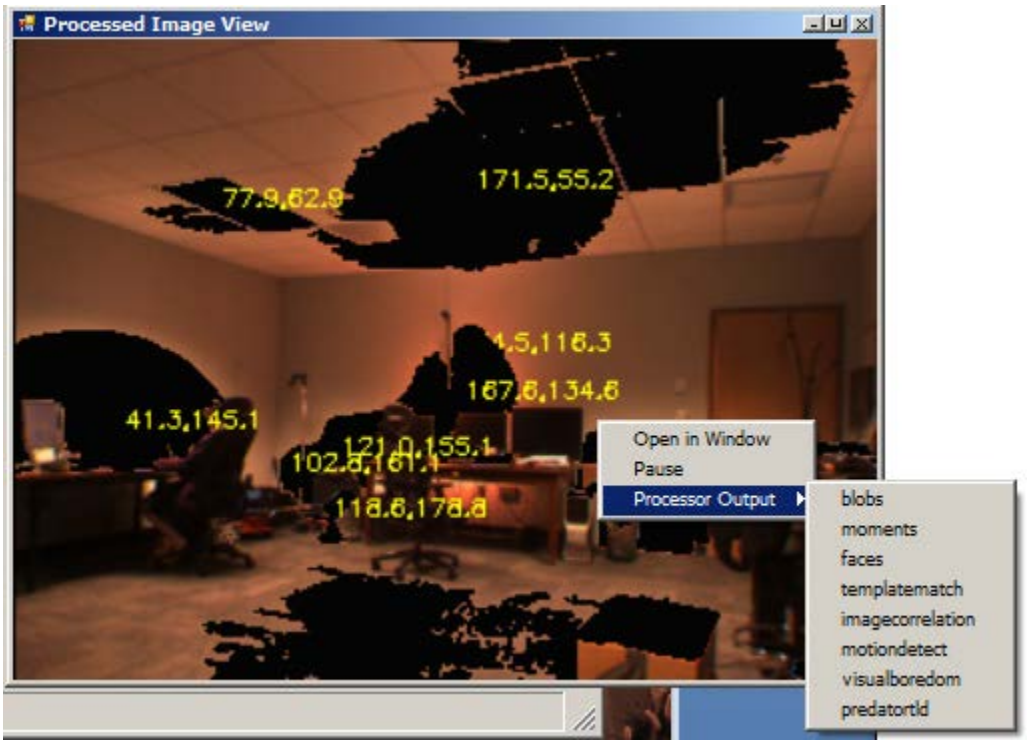
These buttons allow you to manually control the robot's onboard pan–tilt–zoom (PTZ) camera and will override the low-level robot functions used by the symbolic and sub-symbolic system components. Therefore, you should be able to correct the action of the robot's camera even when it is being controlled by the system subcomponents. These controls are only active if SS-RICS is connected to an actual robot fitted with a PTZ camera.

4.1.8 Control Buttons

- U button: when pressed the PTZ camera is moved up.
- D button: when pressed the PTZ camera is moved down.
- L button: when pressed the PTZ camera is turned left.
- R button: when pressed the PTZ camera is turned right.
- H button: when pressed the PTZ camera is returned to its home position.

4.1.9 Camera Views

These display areas show the latest data captures by the PTZ camera attached to the robot. The Raw Image View on the left simply shows the unadulterated images as they are received from the camera. The Processed Image View on the right provides the ability to view the output of the various camera/image-processing subsimprocessors (subsims). This output area contains data that have been superimposed on the image, such as detected motion, blobs, image correlations, face detections, or template matching results. These image processors are discussed later in the manual, and examples of their output are given in each section. The panel may also be detached from the main interface by right-clicking on it and selecting Open in Window. It can be scaled, full-screened, and minimized in this mode. The right-click menu also allows you to select between the outputs of the various camera/image subsims as shown in Fig. 5.



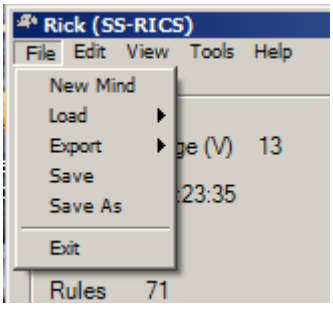
Note: The last item on this menu, predatortld, might or might not exist when you read this. As of this editing, that subsim does not exist. It will exist when we can get it converted from MATLAB to a dll. It is used to track regions of the image as they move and is very robust and accurate. When/if we install this feature, this user's manual will naturally change to reflect that.

Fig. 5 Right-click menu for processed image view panel*

4.1.10 Menus

File Menu

The File menu is shown in Fig. 6.



* In Fig. 5, the last item on the menu, predatortld, might or might not exist when you read this. As of this editing, that subsim does not exist. It will exist when we can get it converted from MATLAB to a dll. It is used to track regions of the image as they move and is very robust and accurate. When/if we install this feature, this user's manual will naturally change to reflect that.

Fig. 6 Main user interface File menu

New Mind

The New Mind menu option creates a new mind and loads it into the system. You will be prompted to save the existing mind to a file if you so choose.

Save and Save As

The Save option will serialize the brain class and its subsimprocessors to a file called Mind.dat (Fig. 7). This file can be loaded using the Mind option discussed previously. The Save As will do the same but also allow you to choose a file name.

Load Submenu

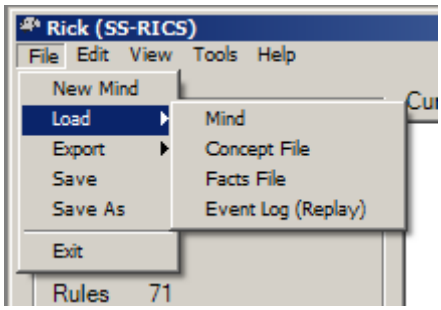


Fig. 7 Load submenu of File menu

Mind

The Mind option will load a specific mind file that you select.

Mind files may be created using the Save or Save As file menu items.

Concept File

The Concept File menu option will load a specific concept file that you select.

Concept files can be hand-created using any text file editor, or you may load any Concept file created for the ConceptNet system. Once loaded you may see an increase in the number of goals and rules in the system depending on the contents of the concept file. There are some example concepts in the concept folder in the SS-RICS directory. After concepts are loaded, they will show up as facts of type "concept" in the Goal Editor (see Section 4.4 on the Goal Edit GUI for more information).

Facts File

The Facts File option will load a specific facts file that you select.

Fact files may be created using the Export submenu item. Currently, it has only one option: Facts.

Event Log (Replay)

The Event Log (Replay) option will load a specific events file that you select.

This file is created automatically as SS-RICS is running and includes all motion directives and goals issued to the robot. The results of the replay can be seen in the simulator and in the views on the BrainInterface panel.

Export Submenu

The Export submenu has one option: Facts (Fig. 8).

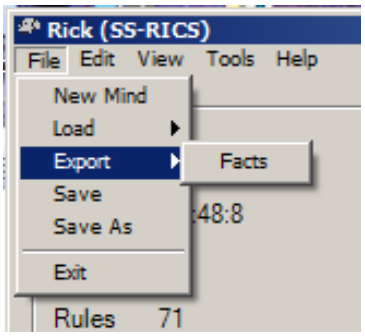


Fig. 8 Export submenu of File menu

The Facts option will save all facts in the system to a file with a name of your choosing (Fig. 9).

Edit Menu

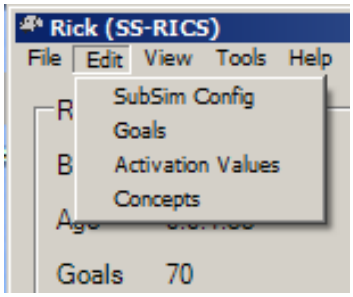


Fig. 9 Main user interface Edit menu

SubSim Config

For more information on the SubSim Config dialog, see Section 4.2 of this manual.

Goals

For more information on the Goal Edit dialog, see Section 4.4 of this manual.

Activation Values

This dialog presents the values used in calculating activation for each current memory type (Fig. 10). The drop-down menu at the top of the screen can be used to select which memory type you want to affect.

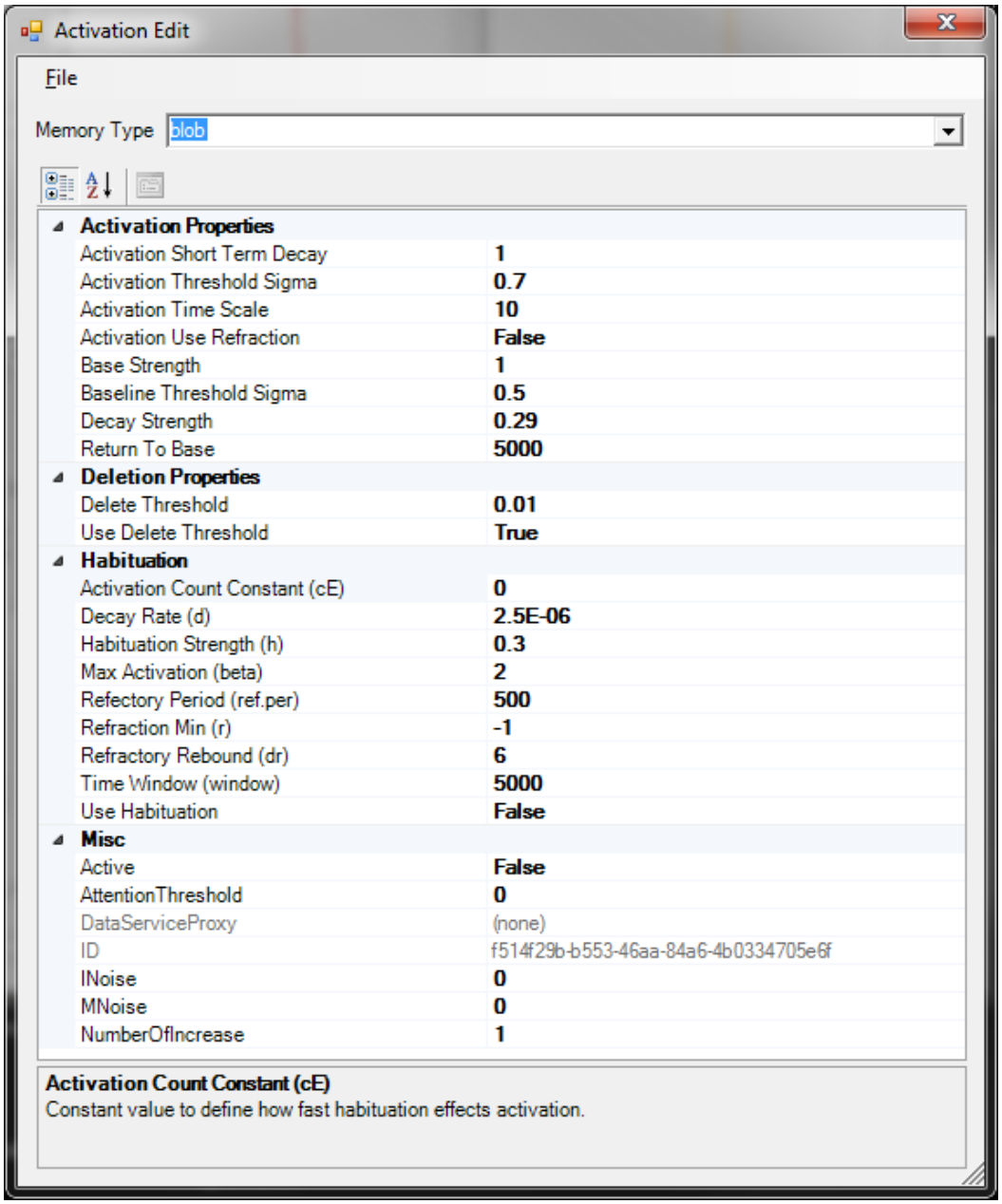


Fig. 10 Activation Properties page

Activation Properties

Activation Short Term Decay
This value defines the decay behavior of refraction.

Activation Threshold Sigma
This value controls the intensity of the noise that gets added to the activation curve. Can be zero for no noise.

Activation Time Scale
This value affects the last time value used for computing the refraction period.

Activation Use Refraction
This value controls the use of refraction, which is an immediate or near-immediate dip in attention once it is activated to simulate the knowledge of having just noticed an event, thus allowing for other memories to be retrieved and not the same one over and over.

Base Strength
This is the strength at which a memory's activation value defaults to.

Baseline Threshold Sigma
This value controls the noise that gets added to the baseline value. Can be set to zero for no noise.

Decay Strength
This value governs the decay rate of a memory.

Return to Base
The number of increases before a memory's activation is reset to baseline.

Deletion Properties

Delete Threshold
This is the value at which a decaying memory is deleted.

Use Delete Threshold
This controls whether or not a memory decays: true means they decay, and false means they do not.

Habituation

Activation Count Constant
Constant value to define how fast habituation effects activation.

Decay Rate (d)
The rate of decay

Habituation Strength (h)
The amount of habituation to apply

Max Activation (beta)
The maximum strength that can be achieved when activation occurs

Refectory Period (ref.per)
The amount of time in milliseconds to refract after activation

Refraction Min (r)
The minimum value to refract to after activation

Refractory Rebound (dr)
The rebound strength during refraction effect refractory rate during refraction

Time Window (window)
The time window in milliseconds to calculate habituation strength using the number of activations in the window

Use Habituation
Turns habituation on and off

Misc

Active
Used by goals and rules to denote whether they are active or not (see Section 4.4 for more information).

Attention Threshold
Activation strength of a goal or rule past which it gains attention

INoise
Obsolete

MNoise
Obsolete

NumberOfIncrease
Number of activations that have occurred; will be compared to Return to Base to determine whether or not a return to the base activation is warranted.

Figure 11 shows the items in the File menu.

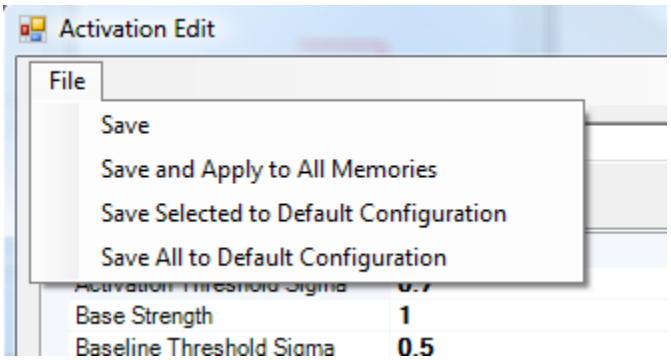


Fig. 11 Activation Properties save options

Save

This saves all the memory types values to the default memory type table in SS-RICS.

Save and Apply to All Memories

This saves all the memory types values to the default memory type table in SS-RICS and sets all active memories' activation values to these.

Save Selected to Default Configuration

This writes the values present for the selected memory type to the config file.

Save All to Default Configuration

This writes the values present for all the current memory types to the config file.

The following is an example of saving settings to the config file:

```
  <Activations>
    <ActivationParameters>
      <add name="FaceCandidate" basestrength="0" decaystrength="0"
returntobasestrength="0" deletethreshold="0" baselinenoisesigma="0"
activationnoisesigma="0" shorttermdecay="1" timescaling="10"
userefraction="False" decays="False" useHabituation="False" dr="6"
h="0.3" cE="0" d="2.5E-06" b="2" r="-1" refPer="500" window="5000" />
    </ActivationParameters>
  </Activations>
```

These values will be read in at startup and will be seen in the Activation Properties window when the user opens it. For a more in-depth description of the config file, see the Configuration File section (14).

Concepts

This option displays the concept editor (Fig. 12).

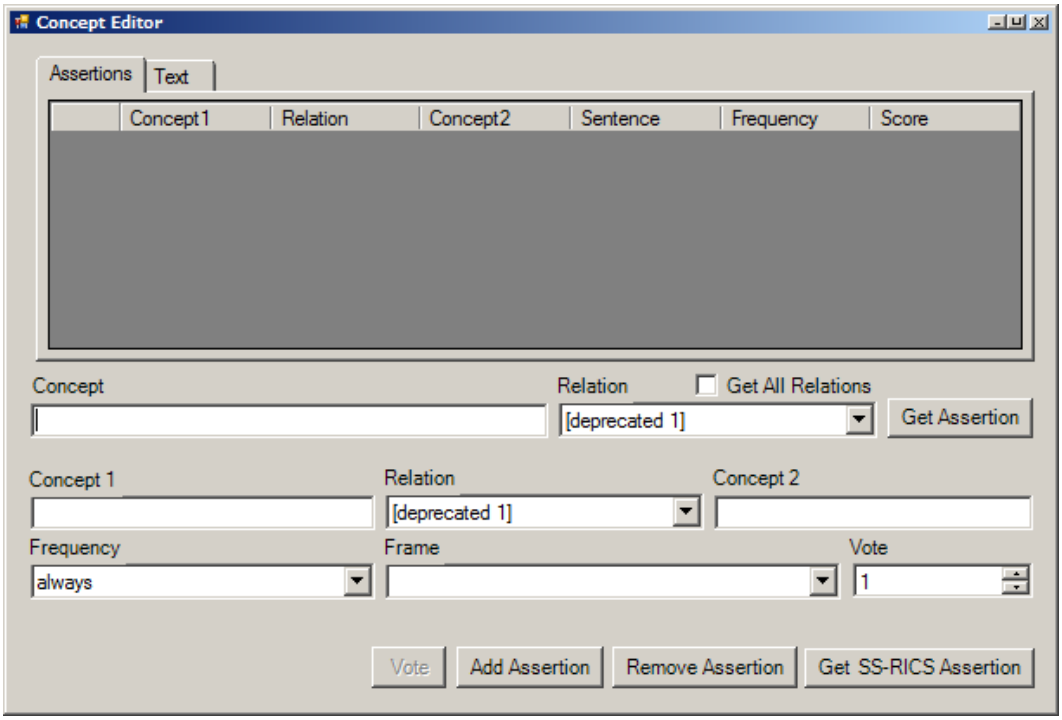


Fig. 12 ConceptNet concept/assertion/relation editor

This interface will allow the user to browse, edit, and remove and add concepts, assertions, and relations in the ConceptNet database that comes with the system.

View Menu

Figure 13 shows the View menu.

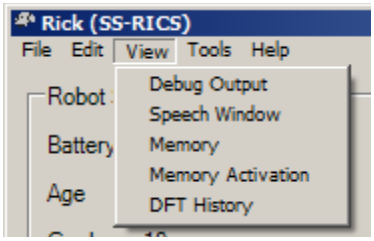


Fig. 13 Main user interface View menu

DebugOutput

Displays a dialog box with 3 panels, which are views of the current status of line and contour determination in the robot.

Line Map Panel

This panel shows the current evolution of the lines being constructed from points detected by the SICK laser (Fig. 14). The Update box, when checked, will show

this process in real time. If not checked and then checked, it will immediately display the current status of line generation. When checked, the Display Names box will display the internal names of each line segment as it grows but will not when not checked. The Display Scale control will, as advertised, increase or decrease the scale of the overall view. The current position of the robot as determined by data received from the wheel encoders is displayed along with the adjusted position using localization.

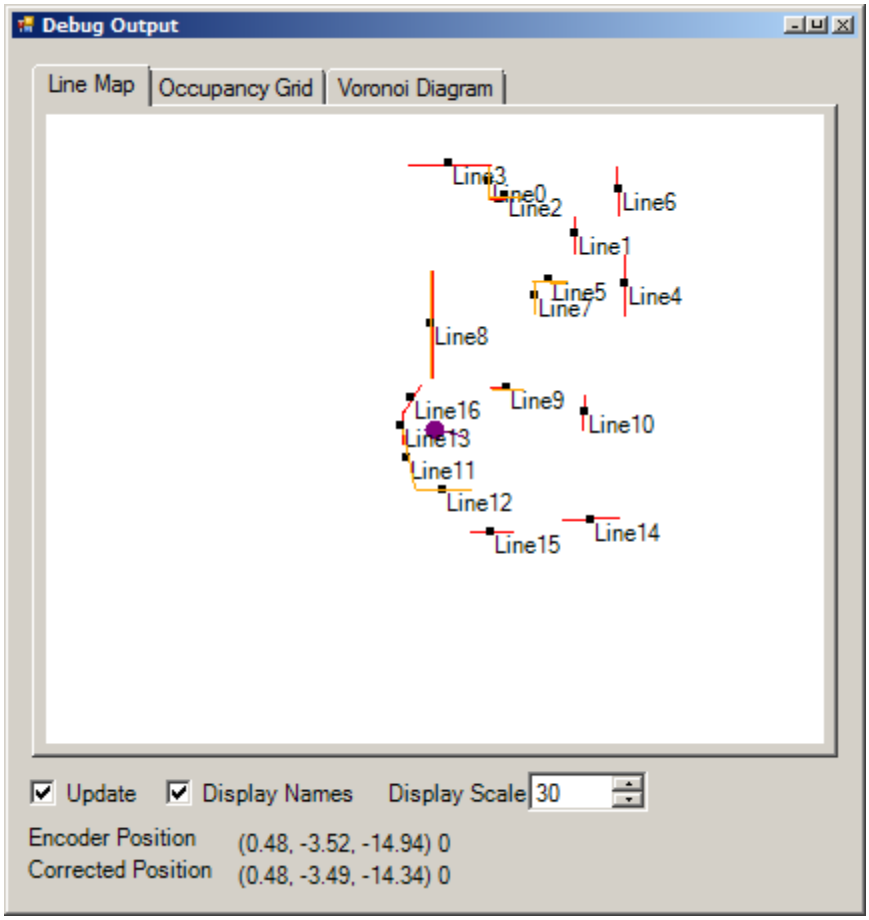


Fig. 14 Debug output dialog, Line Map panel

Occupancy Grid Panel

This panel shows the current population of empty, unknown, and solid object points (Fig. 15). If a map is in use (from using the simulator or the actual robot in conjunction with a map of the area it is in), the grid is generated from a grid file related to the map. If no map is in use, the open-source Karto-Mapper utility is used to generate it. The controls at the bottom do the same thing as the previous panel except show names.

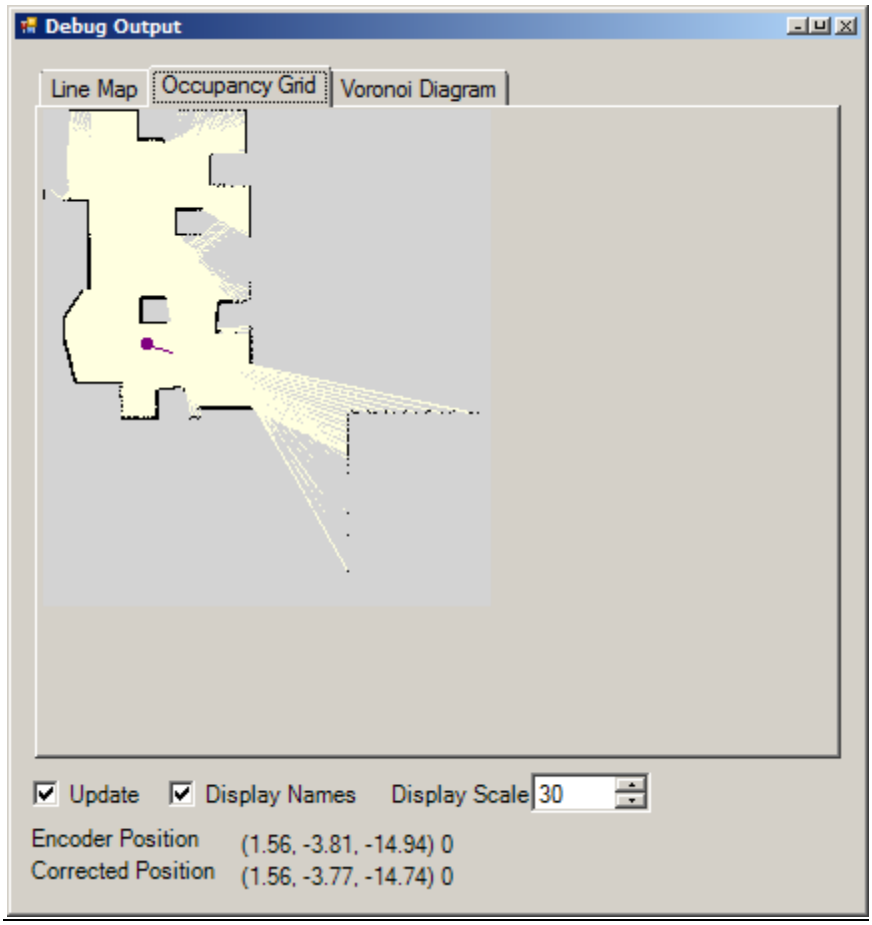


Fig. 15 Debug output dialog, Occupancy Grid panel

Voronoi Diagram Panel

This panel (Fig. 16) is largely for developer debugging and can safely be ignored.

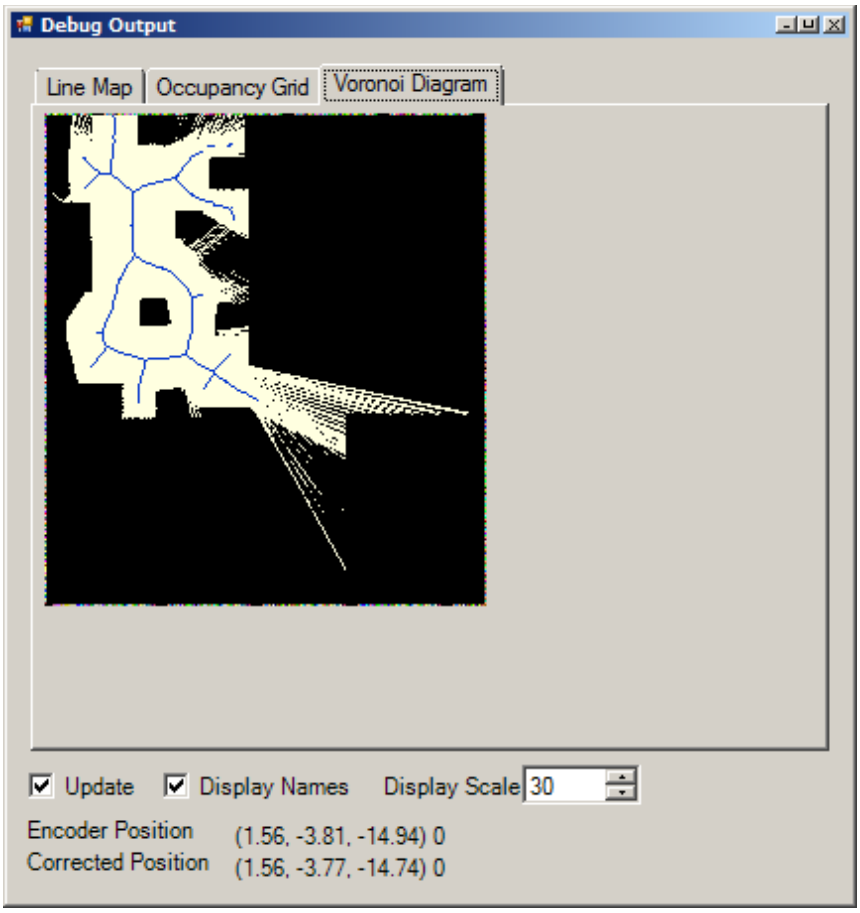


Fig. 16 Debug output dialog, Voronoi diagram panel

SS-RICS Speech Window

Displays a dialog box showing the current status of what the robot is hearing as you speak it.

SS-RICS Speech Window Panel

This panel displays a list of the goals that the system has ready to execute by voice command (Fig. 17). If executing goals in the list is your current primary objective, the Use Dictation Grammar box should be unchecked. If you wish the robot to parse what you are saying using natural language processing (coming soon), this box should be checked. When unchecked, the robot will restrict its matching of what you speak to the list of goals shown in the dialog and will execute them much more quickly and reliably. The Show Debug Data box, when checked, will show what the robot thinks you are saying. This can help identify areas where it has trouble understanding you, and you can then train the speech recognizer offline to perform better in the future.

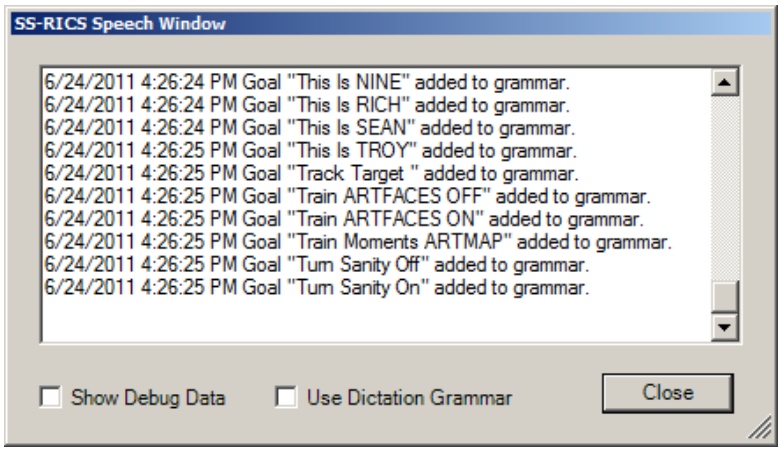


Fig. 17 SS-RICS Speech Window panel

Memory Window

The Memory Window displays a dialog box showing the current memories the robot has accumulated (Fig. 18). The components of each memory are available for viewing by clicking the "+" sign at the beginning of each entry. This is primarily a diagnostic tool but can be useful in normal runs as well. Under the File menu there is an option called Select Font. This will allow you to select a new font and/or size. (Note: Select TrueType fonts only.)

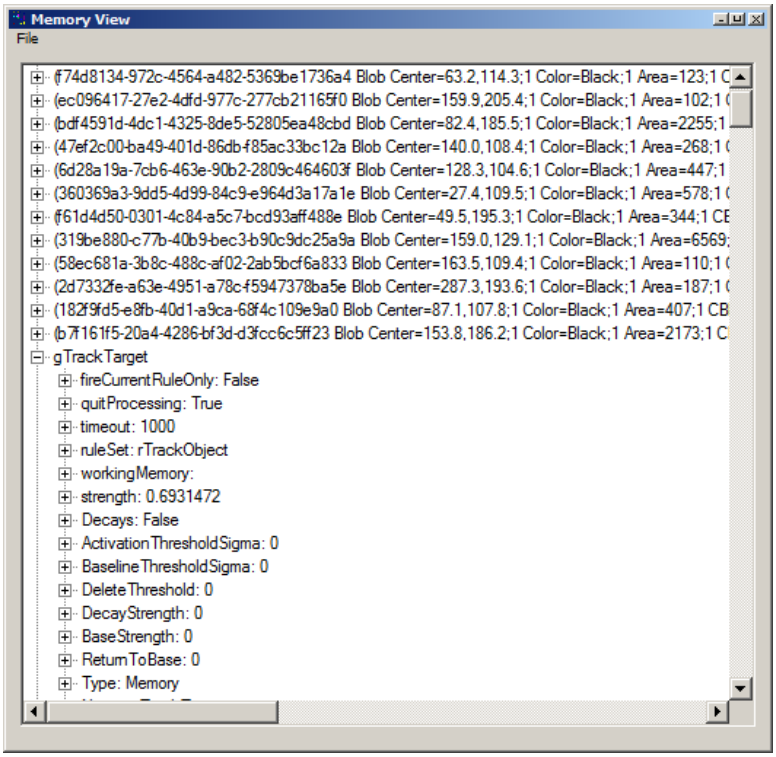


Fig. 18 Robot memory window

Memory Activation Window

This window displays a dialog box showing the current memories and their ongoing activation values. Refer to the Memory Activation Viewer section of this manual for more information.

DFT History Window

Displays a graph showing the weight curves generated by the DFT algorithm in determining a path to follow in a given map (see Fig. 1). The curve showing the highest values (in the case below, the curve colored purple) represents the path chosen out of the possibilities. See Section 2 describing the Choice subsim processor for an example and more detail.

Tools Menu

Figure 19 shows the Tools menu.

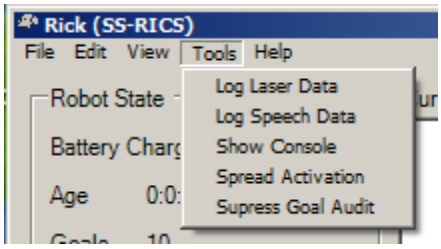


Fig. 19 Main user interface Tools menu

Log Laser Data

When checked, this item writes the laser scan data to a text file.

Log Speech Data

When checked, this item writes the speech debug data (described earlier) to a text file.

Show Console

When checked, this item writes displays a disk operating system (DOS) console window that will display any messages and data written to stdout/stderr. This is primarily for debugging.

Spread Activation

When checked, this will enable the spreading of activation between related memories. This is a method of bringing into play goals and other behavior that is relevant to the current focus of the robot's attention (Anderson and Lebiere 1998).

Suppress Goal Audit

When checked, this will keep all output from goal execution (including those activated via the Attention window) from appearing in the output pane of the Edit Goals window *except* explicit output such as from a print directive and the result of the goal if it was not successfully executed.

Help Menu

Figure 20 shows the Help menu panel.

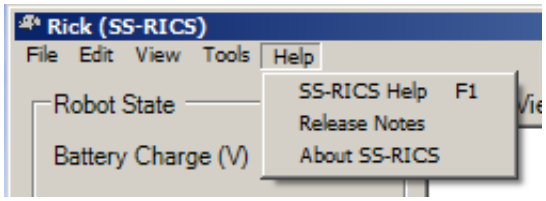


Fig. 20 Main user interface Help menu items

SS-RICS Help

This item brings up the Help facility.

Release Notes

This item displays the current release notes.

About SS-RICS

Figure 21 shows the the About box:

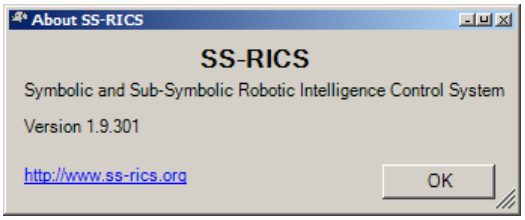


Fig. 21 SS-RICS About box

4.2 SubSim Config Dialog

This interface (Fig. 22) is designed to provide execution and property control of all of the subsimprocessors (subsims) that were loaded for the current run (as described later in the Application Settings section [14.4]).

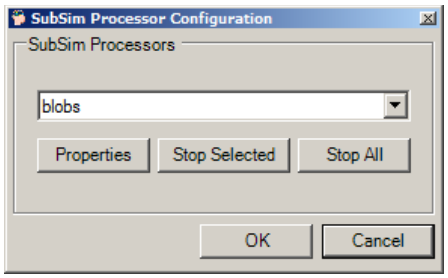


Fig. 22 Subsim configuration dialog

SubSimProcessors

This drop-down menu allows the user to execute the following:

- Start/stop individual subsystems
  - To start/stop a given subsim, select it from the drop-down menu and press Start/Stop Selected. The choice presented is based on whether or not the selected subsim is running or not. If it is running, the button will say Stop Selected, and if not it will say Start Selected. Similarly, the Stop All button means that at least one subsim is running, and Start All means none are currently active.
- Set execution parameters (properties) that control the behavior of the subsim. The current subsims that have properties associated with them include the following:
  - Boredom
  - DataCollection
  - Faces
  - Laser
  - LineSegment
  - Moments
  - Points
  - Visual Boredom
  - Motion Detection
  - Predator
  - Tracking

See Section 9, SubSimProcessor Property Pages, for details.

4.3 Introduction to Goals

Goals (or productions) are scripted actions that can be chained together using constructs called antecedents and consequents. They can communicate with each other using facts created by them and/or the SS-RICS brain, thus providing data to base the direction of an execution path. For instance, a goal can tell the robot to advance until it reaches a certain position and, when it does, have it look for a landmark. It can set a fact to indicate whether or not it found the landmark, and another goal can execute or not based on this information. In the general parlance of the autonomous systems world, this system is known as a control language.

One method of generating goals exists in the SS-RICS interface in the form of dialog boxes that allow the user to construct goals piecemeal using a separate dialog for each element of a goal or rule. While this is useful for small one-off goals, it is rather cumbersome for extended goal chains with complex antecedents and consequents. Thus, a more generalized tool was created to allow users to create goals as one might write code in a text editor with project creation and management, syntax-coloring, cut-and-paste facilities, and real-time error reporting.

4.4 Goal Edit GUI

This dialog (Fig. 23) allows you to view, create, edit, and delete goals, rules, and facts as well as execute and debug the systems goals and rules.

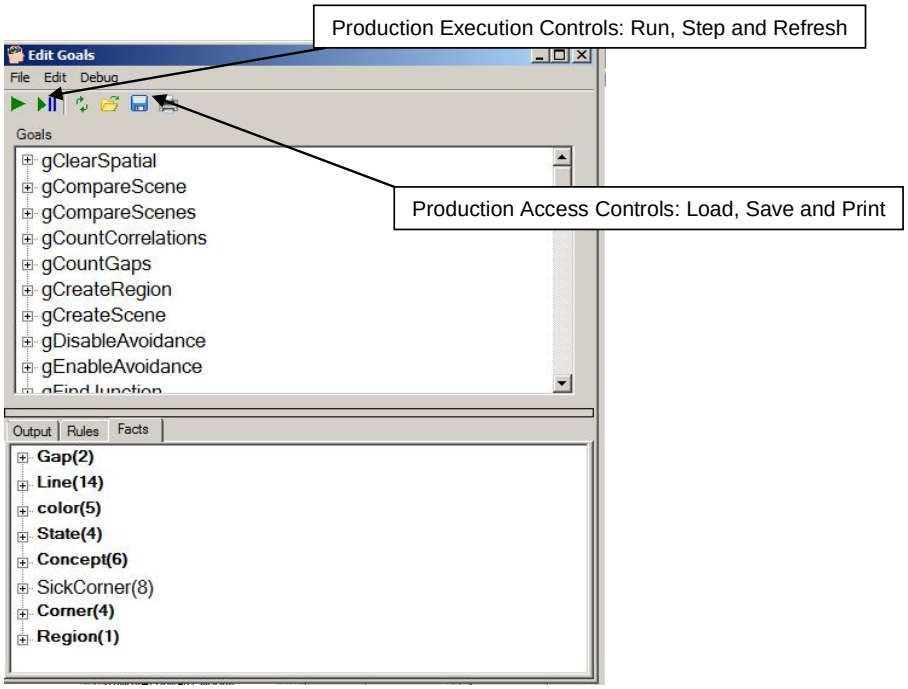


Fig. 23 Goal edit panel controls

The upper portion of the display is dedicated for displaying and editing goals and their corresponding rules. The lower portion is a collection of tabs that allow you to see the systems rules and facts as well as the debug output from any executing goals. The tool bar at the top of the interface gives you direct access to the goal debug/execution commands. The display is not automatically refreshed as memories are modified or changed. To see the latest memories, you must refresh the display by using the tool bar refresh button or the F5 key on your keyboard.

4.4.1 Menus

File Menu

Import

Goal
Imports a goal file into the system. Goal files may be created using the Export menu.

Goals
Import a collection of goals from a goals file. Goals files may be created using the Export menu.

Figure 24 shows the Edit Goals File Import menu.

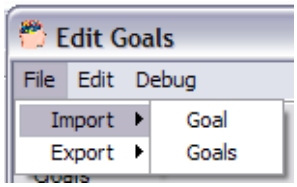


Fig. 24 Goal edit menu, File Import options

Export

Auto Export
Automatically export the goal selected in the goal tree to a file you specify every 15 s.

Goal
Export the goal selected in the goal tree to a file you specify.

Goals
Export all the goals in the goal tree to a file you specify.

Fig. 25 shows the Edit Goals File Export menu.

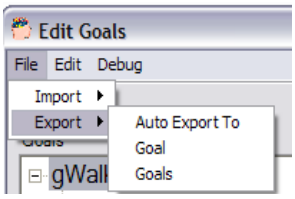


Fig. 25 Goal edit menu, File Export options

Edit Menu

Figure 26 shows the Edit Goals Edit menu.

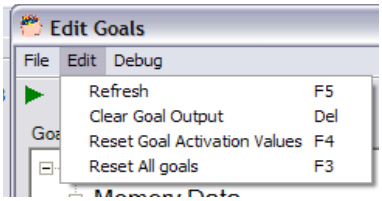


Fig. 26 Goal edit menu, Edit options

Refresh (shortcut key F5)
Refreshes the goals, rules, and facts to match the current set in memory. This is required any time you want to see new rules, goals, or facts that have been automatically generated my SS-RICS's subsystems.

Clear Goal Output (shortcut key Del)
Clears the contents of the goal output window.

Reset Goal Activation Values (shortcut key F4)
Resets the activation values of the goal selected in the goal tree.

Reset All Goals (shortcut key F3)
Resets all the goals in the system. This reset simply makes the goal active so that it may be run by the system.

Debug Menu

Figure 27 shows the Edit Goals Debug menu.

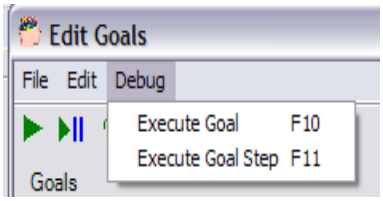


Fig. 27 Goal edit menu, Debug options

Execute Goal (shortcut key F10)
Executes the goal selected in the goal tree.

Execute Goal Step (shortcut key F11)
Executes one step of the goal selected in the goal tree.

4.4.2 Goal Edit Tabs

Output Tab

The output tab (Fig. 28) is designed to show you the output generated by the goal(s) currently executing in the system. (The lowercase "g" shows up in front of the name of the goal to identify it as a goal. Rules will show up appended with a lowercase "r".) Each line in the output window is prefixed with the name of the goal that is outputting the text. This output is designed to be a debut or trace output, so you may see a lot of information displayed. The text is output to the screen from top down so the last item output is always at the top.

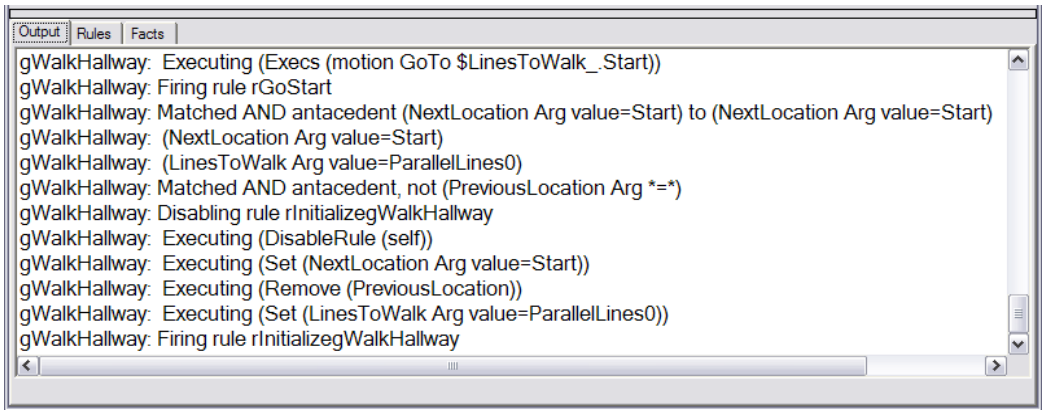


Fig. 28 Goal edit menu, Output tab

Rules Tab

The Rules tab (Fig. 29) is designed to give you hierarchical view of all the rules in the system. Each rule and its corresponding values are displayed any may be modified via a right-click context menu. You may also drag and drop the rules, consequents, and antecedents found in this tab to the goal tree above it.

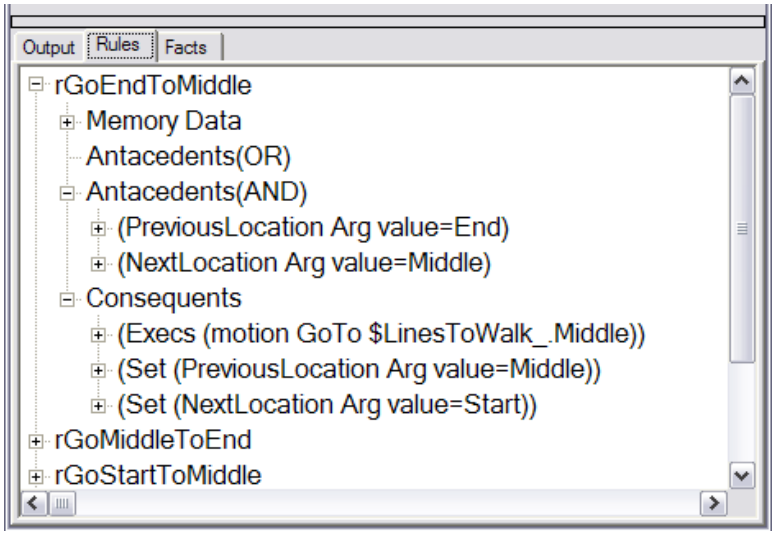


Fig. 29 Goal edit menu, Rules tab

Facts Tab

The Facts tab (Fig. 30) is designed to give you hierarchical view of all the facts in the system. The facts in this view are grouped by type to make the display easer to view. Each fact and its corresponding values are displayed and may be modified via a right-click context menu. The values displayed are not necessarily the values currently in memory, as they represent the values as they were when the interface was loaded or the last time a refresh was performed.

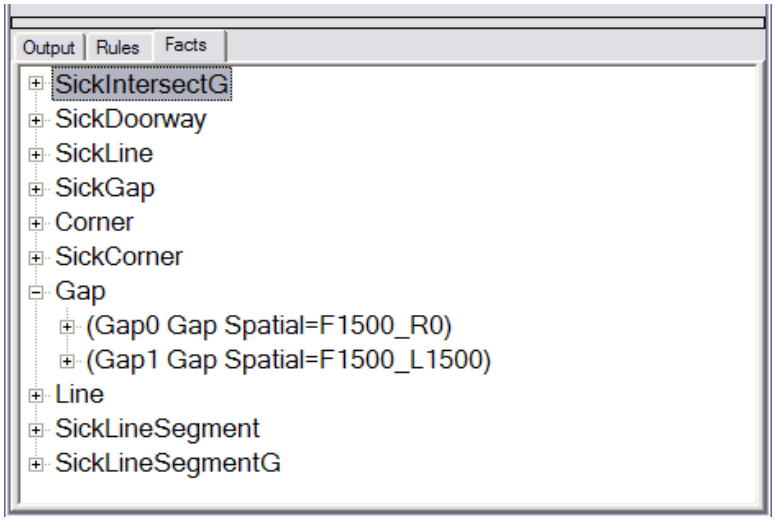


Fig. 30 Goal edit menu, Facts tab

4.4.3 Working with Goals, Rules, and Facts

Goals

A goal is a collection of rules that is executed to perform a logical unit of work.

Adding a new goal

1) Right-click in the white space or on a goal in the goal tree.
2) Select Add Goal from the menu.
3) When the Goal Edit dialog (Fig. 31) appears, you will need to supply at least 2 values: a noun and a verb for the name of the goal. The naming syntax allows machine-learning techniques (substitution) to be applied to the goal.

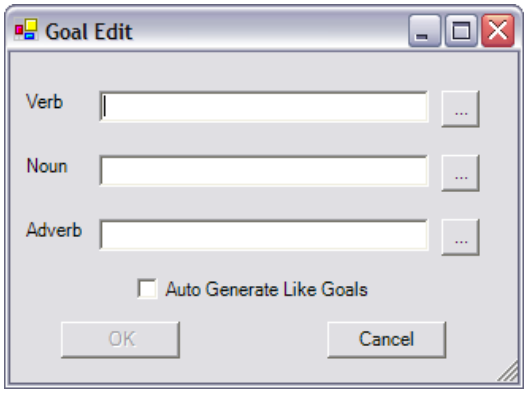


Fig. 31 Goal name edit box

Verb (required)
Verb to be used in the name of the goal. (Note that the verb designation is used later by ConceptNet to build new productions.)

Noun (required)
Noun to be used in the name of the goal. (Note that the noun designation is used later by ConceptNet to build new productions.)

Adverb (optional)
Adverb to be used in the name of the goal.

Editing an existing goal

1) Right-click on the goal you wish to edit.
2) Select Goal Edit from the menu.
3) The Goal Edit dialog (Fig. 32) will appear with the editable properties filled in with the current values.

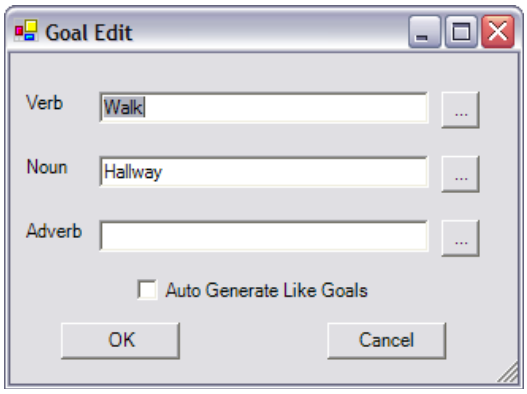


Fig. 32 Example goal name edit box

Rules

A rule is an if–then statement that may contain any number of And or Or antecedents and any number of consequents.

Adding a New Rule

1) Expand the goal node for the goal that you want to add a rule to.
2) Right-click on the Rules node and select Add Rule.
3) The Rule Edit dialog (Fig. 33) will appear, and you will need to supply at least 2 values: a noun and a verb for the name of the rule. The naming syntax allows machine-learning techniques (substitution) to be applied to the rule. Rule names are typically the same as the higher level goal but may be any value you like.

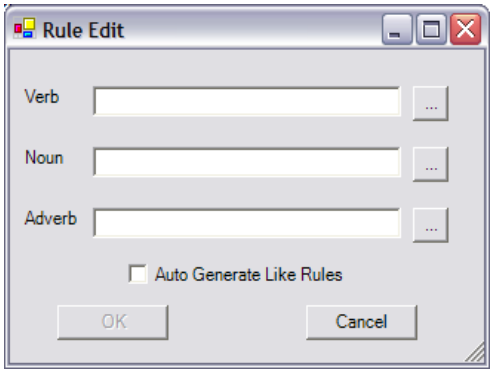


Fig. 33 Rule name edit box

Verb (required)
Verb to be used in the name of the rule.

Noun (required)
Noun to be used in the name of the rule.

Adverb (optional)
Adverb to be used in the name of the rule.

Editing an Existing Rule

1) Expand the goal node for the goal that contains the rule you want to edit.
2) Expand the Rules node.
3) Right-click on the rule node and select Edit Rule.
4) The Goal Edit dialog will appear with the editable properties filled in with the current values.
5) Figure 34 shows the Rule Edit menu.

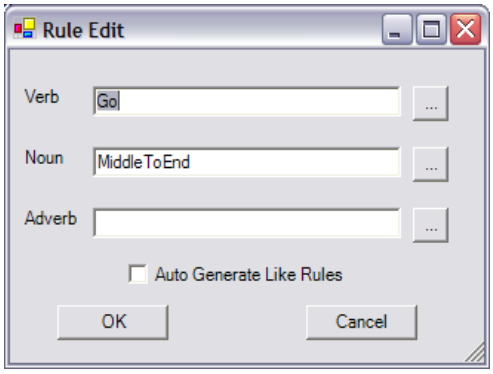


Fig. 34 Example Rule Edit box

Antecedents

An antecedent is the right-hand side. Both the "if" statement of a goal and a goal may have both And or Or antecedents. The And antecedents and the Or antecedents are contained in 2 separate collections that may be modified in the goal tree.

Adding a New Antecedent

1) Expand the rule node for the rule to which you would like to add an antecedent.
2) Right-click on either the Antecedents(OR) or the Antecedents(AND) node and select Add Antecedent.
3) The Fact\Antecedent Edit dialog (Fig. 35) will appear.

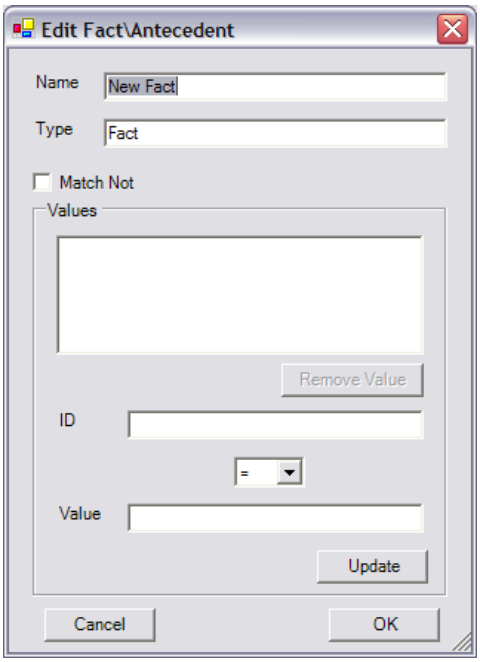


Fig. 35 Fact\Antecedent edit box

Name
The name you would like to use for the antecedent.

This value may be the actual name of the fact you would like to match or it may be a string value using the operators described in the Goal, Rule, and Consequent Operators section (4.4.3) of this manual.

Type
The type you would like to use for the antecedent.

This value may be the actual type of the fact you would like to match or it may be a string value using the operators described in the Goal, Rule, and Consequent operators section (4.4.3) of this manual.

Match Not
When checked, the antecedent will be match when the values provided do not match a fact in memory.

When not checked, the antecedent will be valid when the provided values match a fact in memory.

Values
The values section allows you to add any number if ID=Value pairs to the antecedent. If you wish to match on only one and not all of the slot values in the fact, you may add the * operator to both the ID and Value and add it to the values collection (i.e. *=*).

ID

The ID may be either the actual ID of the slot to be matched or you may provide a string value using the "?" or "$" operators described in the Goal, Rule, and Consequent operators section (4.4.3) of this manual.

Value

The value may be either the actual value of the slot to be matched or you may provide a string value using the operators described in the Goal, Rule, and Consequent operators section (4.4.3) of this manual.

Remove Value button

The Remove Value button removes the selected item from the values collection.

Update button

The Update button adds or updates the values collection with the values provided in the ID and Value text boxes.

Editing an Existing Antecedent

1) Expand either the Antecedents (Or) or the Antecedents (And) node that contains the antecedent you would like to edit.

2) Right-click on the antecedent and select Edit Antecedent.

3) The Goal Edit dialog (Fig. 36) will appear with the editable properties filled in with the current values.

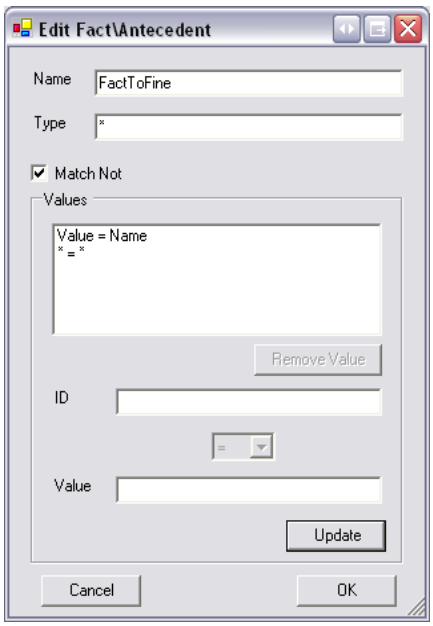


Fig. 36 Example Fact/Antecedent edit box

Deleting an Antecedent

1) Expand either the Antecedents (Or) or the Antecedents (And) node that contains the antecedent you would like to delete.
2) Right-click on the antecedent and select Delete Antecedent.

Consequent

A consequent is the right-hand side. The execution, the "then" of a goal, and a goal may have any number of consequents. SS-RICS has a collection of actions that may be performed in a consequent, and each may use absolute string values or strings using the operators described in the Goal, Rule, and Consequent operators section (4.4.3) of this manual.

Adding a New Consequent

1) Expand the rule node for the rule that you would like to add the new consequent.
2) Right-click on the Consequents node and select Add Consequent.
3) The Edit Consequent dialog (Fig. 37) will appear.

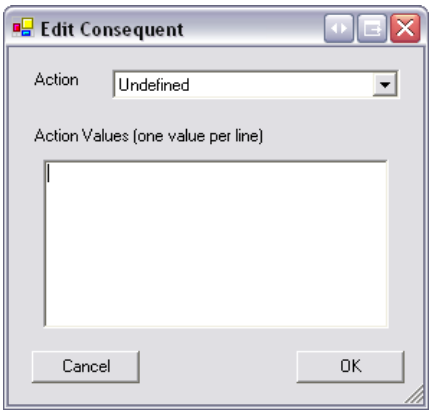


Fig. 37 Consequent edit box

Action

The action that you want the consequence to perform. Refer to the Consequent Actions section (p. 36) of this manual for a detailed description of the available actions.

Action Values

The action values are a collection of values that are to be passed into the selected action. Refer to the Consequent Actions section (page 36) of this manual for a detailed description of the available actions.

Editing an Existing Consequent

1) Expand the rule node for the rule that you would like to edit a consequent to.
2) Expand the consequents node.
3) Right-click on the consequent node that you would like to edit and select Edit Consequent.
4) The Edit Consequent dialog (Fig. 38) will appear with the editable properties filled in with the current values.

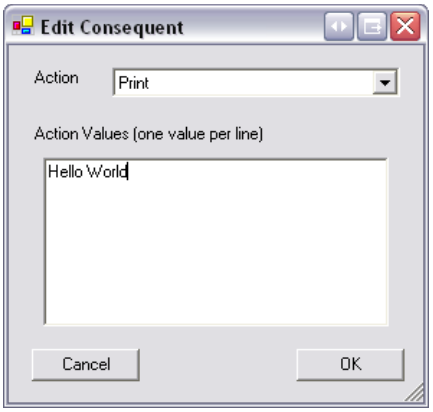


Fig. 38 Example Edit Consequent edit box

Deleting a Consequent

1) Expand the consequents node that contains the consequent you would like to delete.
2) Right-click on the consequent and select Delete Consequent.

Hard-bracketed items are required, while curly braces are optional.

Consequent Actions

Activate

Apply activation to a fact in memory.
Usage:
Activate(factName factType)
Example:
Activate (Corner1 Corner)

Add

Add a fact to memory. If the fact already exists, no action is performed.

Usage:

Add([FactDefinition])

Example:

Add (MyMemory MyType MyValue = 0)

AddRule

Add a rule to the current goal. If the rule does not exist, no action is performed.

Usage:

Add([RuleName])

Ask

Ask the modeler/operator a question. This action is not fully implemented.

Usage:

Ask([Question][FactDefinition])

Example:

(Ask("What would you like to call this scene" SceneName Arg Value))

Decay

Apply decay to a fact in memory. This will call the refraction function on the fact.

Usage:

Decay(factName factType)

Example:

Decay (Corner1 Corner)

DisableGoal

Disable a goal. If the goal does not exist, no action is performed.

Special case:

If you use the keyword "self" as the goal name, the current executing goal is disabled.

Usage:

DisableGoal([GoalName])

DisableRule

Disable a rule. If the rule does not exist, no action is performed. If a rule has been disabled, it will display a "#" before the name of the rule. You can right-click on the rule name to enable it again.

Special case:

If you use the keyword “self” as the rule name, the current executing rule is disabled.

Usage:

Disablerule([RuleName])

Examples:

(DisableRule(self))

(DisableRule(rGreetNewPerson))

EnableGoal

Enable a goal. If the goal does not exist, no action is performed.

Special case:

If you use the keyword “self” as the goal name, the current executing goal is enabled.

Usage:

EnableGoal([GoalName])

EnableRule

Enable a rule. If the rule does not exist, no action is performed.

Special case:

If you use the keyword “self” as the rule name, the current executing rule is enabled.

Usage:

EnableRule([RuleName])

Example:

EnableRule(rImproveCorrelation)

Execa

Execute a call to a sub-symbolic processor or goal in an asynchronous manor (i.e., execute and do not wait for completion before continuing). If the requested sub-symbolic processor does not exist or is disabled, no action is performed.

Special case:

If the keyword “goal” is used in place of the SubsimProcessorName, that goal will be called as a parallel goal.

Usage:

Execa ([SubsimProcessorName] [Command]([Parameters]))

Execa (Goal [GoalName])

Example:

Execa (Goal gCountCorrelations))

Execs

Execute a call to a subsim or goal in a synchronous manor (i.e., execute and wait for completion before continuing). If the requested subsim does not exist or is disabled, no action is performed.

Special case:

If the keyword "goal" is used in place of the SubsimProcessorName, a goal will be executed as a subgoal.

Usage:

Execs([SubsimProcessorName] [Command]([Parameters]))

Execs(Goal [GoalName])

Example:

Execs (Goal gHelloWorld))

Note: For each subsim, there is a special set of 2 commands regarded as keywords for the goal processing system (meaning that no subsim will have a command specific to it that has the same name): startprocessor and stopprocessor.

They perform the same function as the Stop Selected/Start Selected buttons on the Subsim Config panel. Each will suspend the execution of the designated subsim or restart it.

Usage:

Execs([SubsimProcessorName] [startprocessor]))

Example:

Execs (faces startprocessor))

GenerateLikeGoals

This command interfaces with the ConceptNet dataset to generate goals similar to the one specified based on the extracted nouns, verbs, and adverbs contained in the goal.

Usage:

GenerateLikeGoals(goalname)

Example

GenerateLikeGoals(gFindTheDoor) (Note that the lowercase "g" is required.)

This could result in goals like gFindTheDesk, gFindTheHallway, and the like.

GetConcepts

This will get concepts from the ConceptNet database. You must first have a concept to match to.

Example:

```
Goal GetIdea
        {
        Rule GetIdea((target arg value=* *=*)
        {
                (GetConcepts($target HasA 0 0)
        }
        }
```

In this example, the value slot of target is filled with some value (i.e., dog) and is matched to as an antecedent. If the rule matches and fires, the GetConcepts call will then get the forward and reverse assertions for the HasA slot for "dog" in ConceptNet. The (0 0) specify all of the assertions. To limit the retrieval, simply use a number. For example, (1 1) would give you one assertion in each direction.

GetRuleFiredAt

Get the name of the rule fired at the supplied index in the system's temporal memory where the index starts at zero, which represents the action farthest back in time.

Special case:

If a value for OutputFactName is supplied, the name of the rule found is placed in the value slot of a fact with a name equal to the OutputFactName value. If no OutputFactName value is supplied, the name of the rule found is placed in the GetRuleFiredAt fact.

Usage:

GetRuleFiredAt([index] {OutputFactName})

Example:

(GetGoalFiredAt (0))

LoadType

Load a collection of rules into a goals rule collection based on the rule's type property.

Usage:

LoadType([Type])

Print

Print a text value out to the user interface. You can also print the values of variables for debugging.

Usage:

Print([Text])

Example

(Print(Yes))

(Print (?CorrelationsFound.Value)

Quit

Stop executing the current goal.

Usage:

Quit(true)

Rename

Rename a fact from memory. If the fact does not exist, no action is taken.

Usage:

Rename(OldName OldType NewName [NewType])

Example:

(Rename(Corner0 Corner Nook0 Nooks))

Remove

Remove a fact from memory. If the fact does not exist, no action is taken.

Usage:

Remove([Name] [Type])

Example:

(Remove(* Line))

RestartSubSimLog

Restarts the datalogging thread of subsims that log data, such as laser or speech.

Usage:

RestartSubSimLog()

Set

Update slot value(s) of an existing memory. If the memory does not exist, it is added using the supplied values.

Usage:

Set([FactDefinition])

Example:

(Set (SelectedJunction Arg Value=Right))

Undefined

No action defined.

Usage:

Undefined()

Wait

Wait (pause execution) for a specific amount of time in milliseconds.
Usage:
Wait([TimeSpan])
Example:
(Wait (500))

Sub-symbolic Processor Commands

The Execa and Execs consequent actions may use any of the subsims to perform a number of actions. Currently, the motion and line subsims implement the following commands:

Voice Sub-symbolic Processor

Say

Have the robot say something.
Usage:
Say ([sentence])
Example:
Execs (Voice Say "I am happy"))

Boredom Sub-symbolic Processor

The Boredom subsim takes its place in the pantheon of subsims (Templatematch, Motion, Line, Corner, and Voice, to name a few) as a behavior-modeling tool for the concept of habituation. In humans, the need for fresh stimulation and variety is quite strong, and in modeling such behavior in SS-RICS, the concept is implemented by assessing the current laser input as being either stale or fresh. If the robot has been observing the same or similar input for a given number of consecutive samples, it can be said to be "bored" and can report this by creating a system fact of type State and named BoredomLevel. It contains the following 2 slot values:

- BoredomQuotient, which is a numerical value between 0 and 1 where 0 denotes "totally-agog" and 1 can be interpreted as "bored-out-of-skull"
- Bored, which will say either "true" or "false"

Each reading of the laser collected over the time period specified is stored in a matrix, and when the matrix is full, each reading is compared with each other using the Pearson correlation algorithm. If its value is past the value specified by the gradient control, a counter is incremented, and if it exceeds the value specified by the threshold control, the state fact is set to indicate bored. This subsystem's output (the state fact mentioned) can be used as a trigger (when referenced in a goal, for

instance) to get the robot to go wandering around exploring or to get it to perform some other function designated for when it becomes bored.

This process can be fine-tuned by setting values on the property panel for boredom as seen in Section 9.2 Boredom.

Visual Boredom Sub-symbolic Processor

The Visual Boredom subsim is virtually identical to the Boredom subsim except that it takes its reading from consecutive images from the robot's camera. The same correlation algorithm is used, and the same slots and reporting fact are generated of type State but with the name VisualBoredomLevel.

As with the Boredom subsim, this process can be fine-tuned by setting values on the property panel for boredom as seen in Section 9.4 VisualBoredom.

Motion Sub-symbolic Processor

Conceptual Distance Values

Abstract distance values that may be use in some of the following motion commands:

Veryclose: less than 100 mm

Close: less than 1000 mm

Near: less than 2000 mm

Far: less than 5000 mm

Veryfar: more than 5000 mm

Commands

Moveforward

Move forward a given distance in millimeters.

Special Case:

Distance value may be in conceptual distance value.

Using the Center value will make the robot attempt to stay in the center of a room or hallway. In other words, try not to get too close to a wall.

Usage:

MoveForward [distance]

Examples:

Execs(Motion MoveForward 5)

Execs(Motion MoveForward Center 0)

Execs(Motion MoveForward Far)

Movebackward

Move backward a given distance in millimeters.

Special Case:

Distance value may be in conceptual distance value.

Usage:

MoveBackward [distance]

Example:

Execs(Motion MoveBackward 2))

Turnright

Turn to the right so many degrees.

Special Case:

Distance value may be in conceptual distance value.

Usage:

TurnRight [degree]

Example:

Execs (Motion TurnRight 180))

Turnleft

Turn to the left so many degrees.

Special Case:

Distance value may be in conceptual distance value.

Usage:

TurnLeft [degree]

Example:

Execs (Motion TurnLeft 180))

GotoRelative

Go to the an x,y position relative to the robot. The robot's current position is therefore translated to the origin.

Special Case:

Distance value may be in conceptual distance value.

DistanceTo value is optional; specifies how close to get to the spatial location.

Usage:

GoToRelative x,y {DistanceTo}

Example

Execs (Motion GoToRelative x y close)

Goto

Go to the spatial location of a fact in memory. If the fact does not exist or the fact does not contain a spatial slot value, no operation is performed.

Special Case:

Distance value may be in conceptual distance value.

DistanceTo value is optional; specifies how close to get to the spatial location.

Usage:

GoTo [FactName] {DistanceTo}

Example

Execs (Motion GoTo Location1 close)

Absolute Case:

Distance value may be in conceptual distance value.

DistanceTo value is optional; specifies how close to get to the spatial location.

x,y are the absolute x and y coordinates of the destination.

Usage:

GoTo x,y {DistanceTo}

Example

Execs (Motion GoTo x,y close)

Stop

Stop the current motion.

Usage:

Stop

Setlocation

Set the spatial location of a given fact (can be used for waypoint operations).

Special Case:

Slot name is optional. If no slot name is specified, spatial slot is assumed.

Usage:

SetLocation [FactName] {SlotName}

Example

Execs(Motion SetLocation MyStart MyType)

Lookup

Move the camera up a given degree.

Usage:

LookUp [Degree]

Lookdown

Move the camera down a given degree.

Usage:

LookDown [Degree]

Example:

Execs (Motion LookDown 90)

Lookleft

Turn the camera left a given degree.

Usage:

LookLeft [Degree]

Example

Execs (Motion LookLeft 90)

Lookright

Turn the camera right a given degree.

Usage:

LookRight [Degree]

Example

Execs(Motion LookRight 90)

Lookhome

Returns the camera to the home position.

Usage:

LookHome

Example

Execs (Motion LookHome)

Line Sub-symbolic Processor

Createscene

Creates a file for the image or laser correlation algorithm. You can also specify a path name or directory to store your scenes.

Usage:

ImageCorrelation CreateScene [DirectoryName FileName]

Line CreateLaserScene [DirectoryName FileName]

Example

Execs (Line CreateLaserScene SceneName SceneName)

Execs (ImageCorrelation CreateScene SceneName SceneName)

Goal, Rule, Antecedent, and Consequent Operators

*The * Operator*

The * operator is used as an "any value" replacement wild card to signify that any value may be placed in the location the operator is found in a string.

The ? Operator

The ? Operator is used for value retrieval from a matched memory and may be used in any subsequent antecedents or consequents. This operator also may be used in the location postfixed (e.g., MyFact?) and prefixed (e.g., ?MyFact). The prefixed usage defines a variable and assigns it to a matched fact, while the postfixed usage is used for value retrieval. The string ?MyFact is parsed as the name of the fact represented by the MyFact variable. When used in prefix mode, a period may be appended along with a slot name to retrieve a specific slot value from the match memory. For example, ?MyFact.value where ?MyFact is the variable name and Value is the slot name. If the ? operator is used inline with other text and not delimited with spaces, it must be postfixed with a "_" (e.g., ?Line1_Line was found).

The [] Operators

The [] Operators are used to access data in a slot that are in the form of a comma-delimited string like "item1,item2,item3". To access a value by its index, you would use it as follows (for the example given):

?var.values[0] gives the first element "item1"
?var.values[count] will return the number of elements in the list - 3
The array above will be of length count-1

Example:

```
rTestQuestionMark
{
        (Line1? SickLine Direction=north *=*)
=>
        (Print (Found Line ?Line1 at distance ?Line1.Distance and direction
        ?Line1.Direction))
        (Quit())
}
```

This example is matching the variable Line1? to the name of some memory of type SickLine with a direction slot that equals north. Note that then we used the prefixed question mark to print the value (the name of the line in ?Line1). Note also that we can use the dot operator with the variable ?Line1.Distance to find the value of the

distance slot for the name of line that we matched. In this case we already matched to the distance in the antecedent, and we know it is going to be north, but we could have used the * operator to match to distance in the antecedent and then used the ?Line1.Distance call to find the value of the distance slot.

Example:

```
rDetermineTurn
{
        (CurrentLocation? State *=* NextY>?CurrentLocation.Y)
=>
        Execs (Motion TurnRight 90))
        (Quit())
}
```

In the DetermineTurn example, we are matching to a variable named CurrentLocation?, and then we are using that variable to do a comparison with the NextY slot. So, we are getting the name of some memory of type State, then using that name and the value of the Y slot to compare with the NextY slot. So, the ?CurrentLocation.Y is doing a comparison with the slot value NextY.

The $ Operator

The $ operator is a shorthand notation for a specific fact that may be used as a variable in both antecedents and consequents. The fact type must be Arg. When the $ is placed before the name of a fact that is of type Arg and that fact has a value slot, the value found in the slot is used as a variable in an antecedent or consequent. If the $ operator is used inline with other text values, it must be post fixed with a "_" to designate the end of the variable (e.g., $Line1_memory was found).

Example:

```
rTestDollarSign
{
        (* $LookType Distance=$LookDistance Direction=$LookDirection *=*)
=>
        Print(Found Type:$LookType Direction:$LookDirection)
}
```

The TestDollarSign Rule shown matches on the type and direction found in the following facts:

```
(LookType Arg Value=…)
(LookDistance Arg Value=…)
(LookDirection Arg Value=…)
```

Then it prints out that a fact with the values in the $ facts was found and uses the values from the $ facts to show the match criteria.

Mathematical Expression Evaluation

SS-RICS supports more than 16 operators and 30 functions. Boolean, trigonometric, and numerical approximation functions are available. All mathematical expressions must be wrapped in special start and end operators and may *not* contain spaces. To start a mathematic expression, use the opening operator <%, and to end the expression, use the closing operator %>. Within the expression, open-and-close braces, [ ], must be used in place of open-and-close parentheses, ( ) because the SS-RICS parsing engine already uses parentheses for fact, antecedent, and consequent declarations. Along with the mathematical operators, you may also use the standard goal, rule, antecedent, and consequent operators with the exception of the wildcard operator *, as it is reserved for multiplication operations. The following collection of examples, followed by lists of the available operators (Table 1) and functions (Table 2), may be used in any logical combination within a mathematical expression.

Examples of valid expressions that may be used within rules, facts, antecedents, and consequents:

Add constant values 5 and 2

Expression: <%5+2%>

Result: 7

Add variables in working memory using the ? operator:

Expression: <%?MyNumber.Value_+5%>

Result: If the value slot of the memory matched to ?MyNumber is 2, the result will be 7. If the value slot does not exist or the value is not numeric, the result is Unknown_Math_Parse_Error. Note the underscore "_" at the end of the variable "?MyNumber.Value" to designate the end of the variable name.

Examples (as expressed in the goal editor):

```
Operator "+"
goal IncrementMemory
{
      rule IncrementMemory(
      (MyMemory MyType MyID=*;1)
      {
      Set(MyMemory MyType MyID=<%?MyMemory.MyID_+1%>)
      Quit (True)
```

```
	}
}
```

The following example is very useful for incrementing a counter in memory:

```
Function “rand[min,max]”
goal IncrementRandom
{
	rule IncrementMemory(
	(MyMemory MyType MyID=*;1)
	{
	Set(MyMemory MyType MyID=<%?MyMemory.MyID_+rand[0,1]%>)
	Quit (True)
	}
}
```

Table 1 Math expression operators

Operator	Usage	Description
+	x+y	Add 2 numbers
+	+x	Unary addition
-	x-y	Subtract 2 numbers
-	-x	Unary minus
*	x*y	Multiply 2 numbers
/	x/y	Divide 2 numbers
^	x^y	Raise x to the power of y
%	x%y	Modulo; find remainder of the division of x by y
>	x>y	Greater than logical operator. Returns 1 if true, 0 if false.
<	x<y	Lesser than logical operator. Returns 1 if true, 0 if false.
>=	x>=y	Greater than or equal logical operator. Returns 1 if true, 0 if false.
<=	x<=y	Lesser than or equal logical operator. Returns 1 if true, 0 if false.
!=	x!=y	Not equal logical operator. Returns 1 if true, 0 if false.
==	x==y	Equal logical operator. Returns 1 if true, 0 if false.
&	x&y	And logical operator. Returns 1 if true, 0 if false.
|	x|y	Or logical operator. Returns 1 if true, 0 if false.
!	x	Not logical operator. Returns 1 if x is 0, else returns 1

Table 2 Math expression functions

Function	Usage	Description
abs	abs[x]	Absolute value
acos	acos[x]	Arccosine
asin	asin[x]	Arcsine
atan	atan[x]	Arctangent
avg	avg[x,y,z,...]	Average of a set of values
bin	bin[n]	Converts a binary number to a decimal number
ceil	ceil[x]	Round up to the nearest integer
cos	cos[x]	Cosine
cosh	cosh[x]	Hyperbolic cosine
fact	fact[x]	Factorial of x: x*(x-1)*(x-2)*...(x-x+1)
floor	floor[x]	Round to the bottom
hex	hex[n]	Converts an hexadecimal number to a decimal number
if	if[condition, val1, val2]	If the condition is true, returns val1, else returns val2.
isFinite	isFinite[x]	Returns 1 if the number is finite. A NaN value is not finite.
isNaN	isNaN[x]	Returns 1 if the number is a NaN value
log	log[x]	Natural logarithm
log10	log10[x]	Base-10 logarithm
max2	max[x,y]	Maximum of 2 values
max3	max[x,y,z]	Maximum of 3 values
maxUndef	maxUndef[x,y,z,...]	Maximum of a set of values
min2	min[x,y]	Minimum of 2 values
min3	min[x,y,z]	Minimum of 3 values
minUndef	minUndef[x,y,z,...]	Minimum of a set of values
rand0	rand[]	Random value between 0 and 1
rand2	rand[min,max]	Random value between min and max
round	round[x]	Round to the nearest integer
sin	sin[x]	Sine
sinh	sinh[x]	Hyperbolic sine
sqrt	sqrt[x]	Square root
sum	sum[x,y,z,...]	Sum all values
tan	tan[x]	Tangent
tanh	tanh[x]	Hyperbolic tangent

Note: The functions max2, max3, min2, min3, rand0, and rand2 must not be coded with the suffixed numbers. Use max, min, or rand and include the apropos arguments as described in Usage column.

Facts

The Facts tab at the bottom of the Goal Edit interface allows you to modify the facts contained in SS-RICS's memory.

Adding a New Fact

To add a new fact, select the Facts tab (Fig 39) and right-click on an existing fact or the empty white space. From the menu, select Add Fact and a new fact will be added to the facts list (Fig. 40).

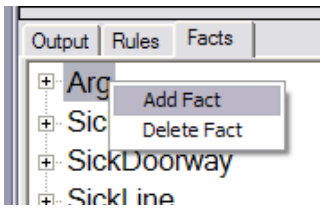


Fig. 39 Fact tab options

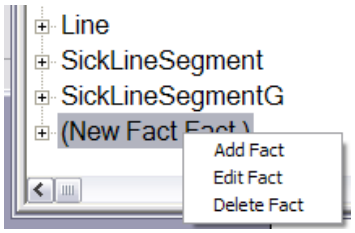


Fig. 40 Fact tab, new fact options

Once the new fact is added, right-click on the fact and select Edit Fact and the Edit Fact\Antecedent dialog will be displayed (Fig. 41).

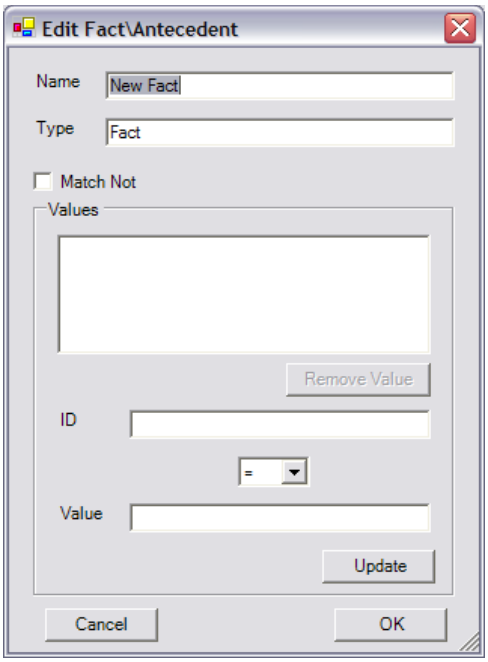


Fig. 41 Fact\Antecedent edit box

Name
The name you would like to use for fact.

Type
The type you would like to use for the fact.

Match Not
This checkbox is not applicable for when adding or editing facts.

Values
The values section allows you to add any number if ID=Value pairs to the fact.

ID
The ID value to be used for the slot.

Value
The value to be used for the slot.

Remove Value button
The Remove Value button removes the selected item from the values collection.

Update button
The Update button adds or updates the values collection with the values provided in the ID and Value text boxes.

Editing an Existing Fact
Right-click on the fact you wish to edit and select Edit Fact and the Edit Fact\Antecedent dialog will be displayed. Update the fact using the dialog as described in the Add Fact section and click OK.

Deleting a Fact
Right-click on the fact you wish to delete and select Delete Fact.

4.4.4 Debugging/Executing a Goal

Play

The Play toolbar button, Execute Goal menu option, and the F10 key can all be used to execute the goal selected in the goal tree. When the goal is executed, its output will be visible in the Output tab.

Step

The Play/Pause toolbar button, Execute Goal Step menu option, and the F11 key can all be used to execute one step in the goal selected in the goal tree. When the goal is executed, its output will be visible in the Output tab.

Stop

The Stop toolbar button is used to stop the execution of the currently running goal.

Refresh

The Refresh toolbar button, the Refresh menu option, and the F5 key all can be used to refresh the data in the window to reflect the system's current memory.

4.5 Goal Text Editor

This section describes how to create, edit, and deploy goals using the text editor. The syntax of this language is of most importance and will be discussed extensively. It is expected that subtle nuances like the usage of "$" or "?", how to formulate goals and rules, and the nature and purpose of antecedents and consequents are well understood by the user. Therefore, we will concentrate on how to construct them using the goal editor syntax and grammar.

In general, a goal consists of one or more attributes and/or rules. A rule contains zero or more antecedents (pre-existing conditions with an and/or flexibility) and

one or more consequents (actions taken based on the validity of the aforementioned antecedents). Rules and consequents may also have attributes. Attributes are metadata or tag-data that further document and control goal execution. Knowledge of attributes as well as antecedent (fact) structure is assumed as well.

4.5.1 Creating a New Project

To create a new project, open Visual Studio (VS) and click on the file menu, then select New and then Project. This will bring up a dialog box with a list of available project types in the left-hand box of the dialog.

Select SS-RICS Goal Project, give it a name, and hit Enter. A sample goal that prints “Hello World” will be generated as a starting point and shown ready to edit in VS. This file will contain the following code:

```
/*
 * Goals
 */

 goal SayHello
 {
      rule SayHello()
      {
            Print( "Hello World" )
            Quit( True )
      }
 }
```

It will be stored in a file named goals.rbt, which is essentially a text file. Any number of goals can be coded in this file, and any number of files of this type can be added to the project by performing a right-click on the project in the solution explorer, which is normally to the right of the text window. This will allow you to create a new file or load an existing one. When it is time to compile the project, any of the files can be excluded by changing their Build Action from Compile to Content or None and, conversely, they can be reactivated by resetting the Build Action to Compile (Fig. 42).

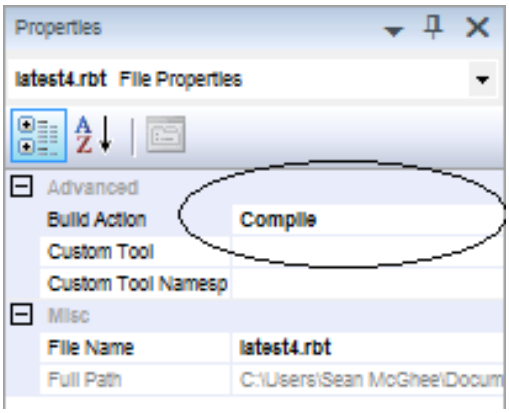


Fig. 42 Build Action properties panel

4.5.2 Opening an Existing Project

To open an existing project select the File menu, then Open, then select Project/Solution and choose a project to open.

4.5.3 Editing and Syntax

The syntax of a goals file is quite similar to "C" language but without the semicolon statement separator.

4.5.4 Elements of a Goal File

Goal files consist of the following bits of grammar:

- Comments
- Goal blocks
- Rule blocks
- Antecedents
- Consequents
- Attributes

Comments

Comments take the form of comments in the "C" language and can be inserted virtually anywhere in the file (meaning we have not found an example that causes a syntax or grammar error). Comments look like the following:

```
/*
 * Goals
 */
```

or

```
/* Goals */
```

As in “C”, they are ignored by the compiler.

Goal Blocks

Goal blocks have the following syntax:

```
goal GoalTest
{
      <rule blocks>
}
```

There must be at least one goal block in the file. They cannot be nested and usually contain (for them to justify taking up space in the file) a rule to be fired. The keyword “goal” is followed by the name of the goal (GoalTest), which must adhere to the camelback naming convention. The first letter of each word in the goal name (must be at least 2 English words roughly corresponding to verb and noun and no embedded spaces) must be capitalized as such: FindTheDoor or GoToStartingPosition. The goal name must also be unique to the project, meaning that not only must you not have the same goal name in the file you are editing, but you also may not have it in any other file that is a part of the project, as they all get compiled together.

Goal blocks usually contain rules to be evaluated and fired. The grammar allows for no rules to be present, but this is not desirable, as the point of the goal would be moot. A goal can be thought of as an overall objective, and rules are the steps required to fulfill that objective.

Rule Blocks

Rule blocks have the following syntax:

```
rule RuleTest1( <antecedents> )
{
      <consequents>
}
```

Rule blocks can contain antecedents and/or consequents. Antecedents define the conditions under which the consequents get executed.

Antecedents

Antecedents are conditions that allow the execution of a rule or not. The antecedent section functions like an if-statement in that it evaluates the components logically to end up with a true or false condition. It is structured using “||” and “&&” for Or

and And operators to separate the conditions and the "!" for logical Not. Thus, the antecedent block

```
rule RuleTest1( (ant1) || (ant2) && (ant3) && !(ant4) )
{
      <consequents>
}
```

can be read as

```
      if( (ant1) || (ant2) && (ant3) && !(ant4) ) perform
consequents
```

Antecedents can be of 2 types: facts or rule names. Antecedents of the type fact refer to facts in existence in the brain/mind. Fact antecedents have the following syntax:

```
(name type slot=val;1…)
!(name type slot=val;1…)
```

In this case, "name" is the name of the fact; "type" is the type designation of the fact, and "slot=val;1…" indicates zero or more slot/value pairs may be present. The ";1" is a value tacked on to the end of each slot to indicate a weighting that is to be applied to the slot from ConceptNet. This is currently not being used. Each antecedent must be enclosed in parentheses. A fact-antecedent evaluates to true if there is a fact in memory that matches it. A fact-antecedent with an "!" in front of it evaluates to true if there is no fact in memory that matches it.

A rule-antecedent is the name of an existing rule that evaluates to true if that rule's antecedent block evaluates to true. As with fact-antecedents, the "!" symbol before the name of the rule-antecedent means that the antecedent evaluates to true if the rule's antecedents evaluate to false. Rule-antecedents have the following syntax:

```
(rulename)
!(rulename)
```

Note that a rule-antecedent consists only of a rule name enclosed in parentheses. A fact-antecedent contains at least the name and type enclosed in parentheses.

The following is an example of a rule block with multiple antecedents:

```
rule LoadPath(
            (Paths Choice *=*;1) &&
            (?Paths.SelectedFact Path FirstLeg=*;1 *=*;1) &&
            !(ValidPlan Arg Value=True;1)
            )
      {
            …
      }
```

Note that the elements of each antecedent in the example parse correctly. If a squiggly red line appears under one of the elements, it may be necessary to quote the element using double quotes, as in the following example of an antecedent (Fig. 43).

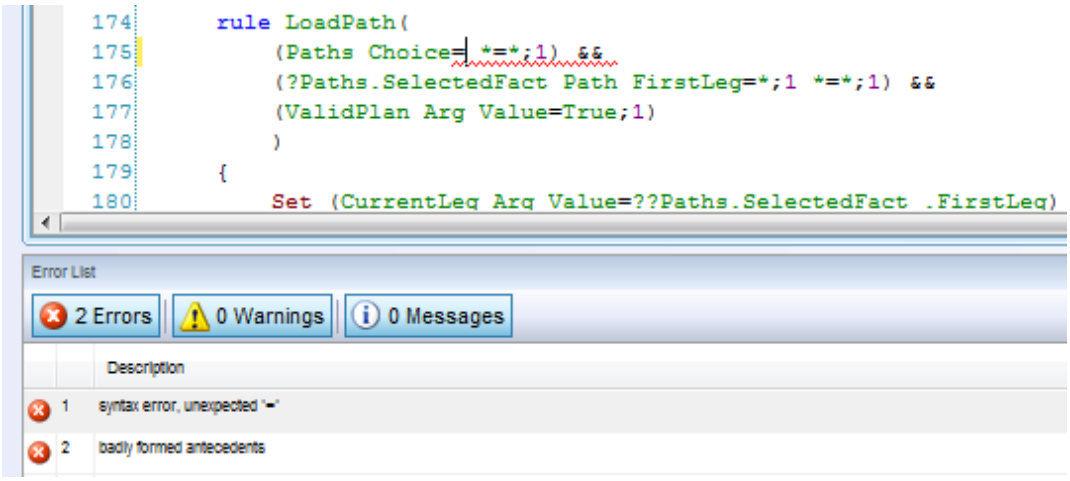


Fig. 43 Error tab with errors

If you wished the type of the antecedent fact to be Choice=, you would then put quotes around it, as shown in Fig. 44.

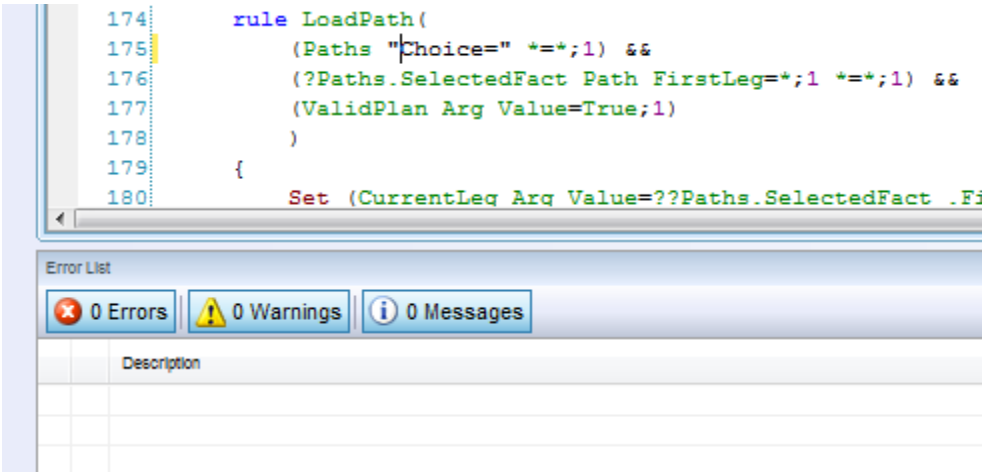


Fig. 44 Error tab after errors corrected

This strategy will work for most syntax errors of this nature. For a more thorough discussion of the nuances of Goal Text Editor, Section 4.5.

Consequents

Consequents are the actions taken by the rule when the antecedent block evaluates to true. If there is no antecedent block, the rule is always executed. The following is an example of consequents in a rule block:

```
      rule FollowPath(
            (CurrentLeg Arg *=*;1 Value!=Null;1) &&
            (?CurrentLeg.Value PathLeg NextLeg!=Null;1 *=*;1)
            )
      {
            Execs (Voice Say Going to
??CurrentLeg.Value_.Endregion_)
            Print (Going to ??CurrentLeg.Value_.Endregion_
??CurrentLeg.Value_.EndPoint_)
            Execs (Motion GoTo ??CurrentLeg.Value_.EndPoint
Close)
            Execs (Motion SetLocation
??CurrentLeg.Value_.Endregion Region Self)
            Set (??CurrentLeg.Value_.StartRegion Region
Self=@Remove)
            Set (CurrentLeg Arg
Value=??CurrentLeg.Value_.NextLeg)
            Quit (True)
}
```

As with antecedents, consequents use the space character to delimit factors within parentheses. Unlike antecedents, consequent declarations have a slightly different syntax, as follows:

```
<Action> (<factors>)
```

"Action" is any of several directives that can be issued and "factors" are the parameters specific to each directive. This example includes the commands Execs (execute the command synchronously), Print (print a text value to the user interface), Set (change the slot values of an existing fact in memory), and Quit (stop the rule from firing again).

The following is an example of a goal with more than one rule, multiple antecedents including a rule-antecedent, and multiple consequents:

```
goal FollowPath
{
      rule FollowPath(
            (CurrentLeg Arg *=*;1 Value!=Null;1) &&
            (?CurrentLeg.Value PathLeg NextLeg!=Null;1 *=*;1)
            )
      {
            Execs (Voice Say Going to
??CurrentLeg.Value_.Endregion_)
            Print (Going to ??CurrentLeg.Value_.Endregion_
??CurrentLeg.Value_.EndPoint_)
```

```
            Execs (Motion GoTo ??CurrentLeg.Value_.EndPoint
Close)
            Execs (Motion SetLocation
??CurrentLeg.Value_.Endregion Region Self)
            Set (??CurrentLeg.Value_.StartRegion Region
Self=@Remove)
            Set (CurrentLeg Arg
Value=??CurrentLeg.Value_.NextLeg)
            Quit (True)
      }
      rule NoPath(
            (rFollowPath  )
            )
      {
            Quit (False)
      }
}
```

4.5.5 Editing Tools

The goal compiler features IntelliSense support, where in certain sections of the code you can simply type and a drop-down list of available choices will appear. For instance, if you are inside a rule block (not the antecedent block at the top of it but the section where consequents are coded), just entering a character will cause the drop-down to appear, as shown in Fig. 45.

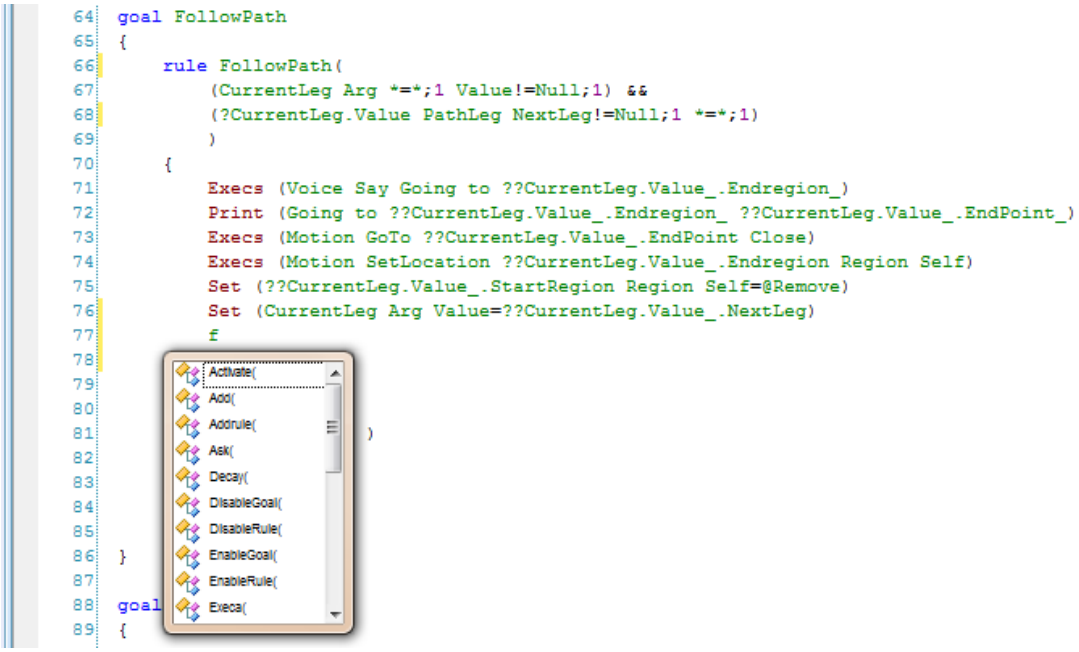


Fig. 45 IntelliSense drop-down for consequents, no match

Note that the letter “f” was typed, and the first entry of the list was indexed. This is because no entries in the list begin with an “f”. If the letter typed was the first letter

of a command in the list, the list would display the list with the first entry beginning with that letter indexed, as shown in Fig. 46.

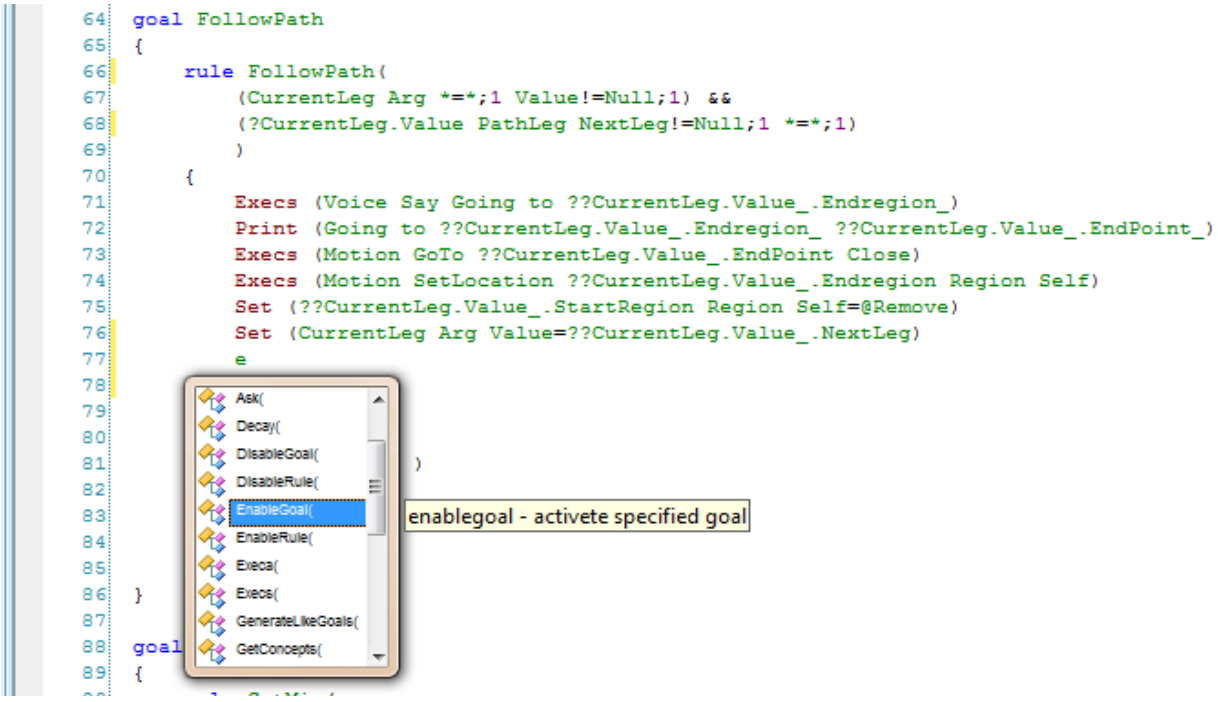


Fig. 46 IntelliSense for consequents, match found

As shown in Fig. 46, when the selection you want is highlighted, a tool tip appears next to it with a brief explanation of the command.

Another place where a drop-down IntelliSense menu can be used is in the antecedent section. When coding antecedents, the name factor (described previously in the antecedents paragraph) is collected and kept in a list. It is often useful to refer to these names repeatedly in a goal, and so when coding an antecedent, one can enter a "?", as shown in Fig. 47.

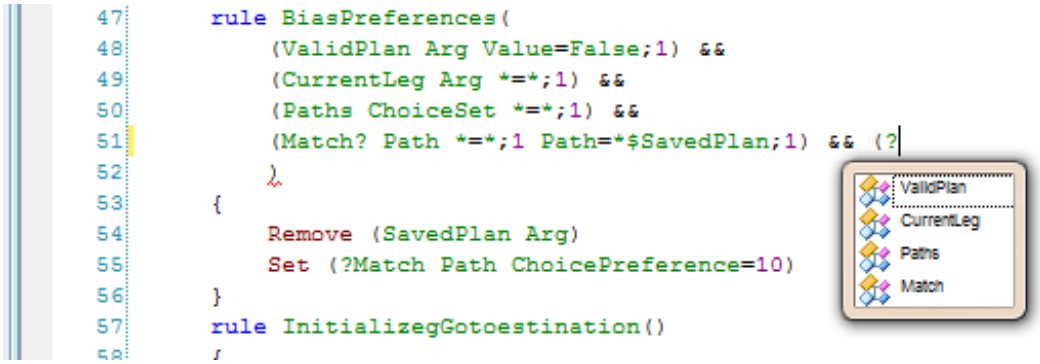


Fig. 47 IntelliSense drop-down of known antecedent names

As shown, the rule's antecedents contain the same list of name fields as the drop-down. For this to work, the new antecedent being coded must not have the closing parenthesis present. The context detection for when, where, and what to display in an IntelliSense menu is a bit tricky, so they may be displayed in other portions of the code where not applicable. Simply ignore them.

4.5.6 Attributes

Attributes are metadata and ancillary information that can affect the behavior of goals, rules, and consequents. An attribute is known by its position in the code and by the syntax using [" and "] as delimiters.

Goal attributes are placed immediately before the goal block they are to apply to, as follows:

```
[Slots(comments="comment for goal GoalTest" verb=Change
noun=GoalName adverb=Rudely)]
[Activation(returntobase=11 basestrength=22.1 decaystrength=33.1
deletethreshold=44.1 baselinethresholdsigma=55.1
activationthresholdsigma=66.1)]
[Active(val=true )]
[Decays(val=true )]
goal GoalTest
 {
```

As shown, there are numerous attributes that can be assigned to a goal. The syntax is straightforward and without much variance.

Similarly, rule attributes have the same syntax:

```
[Active(val=false )]
[Decays(val=true )]
rule RuleTest3( ( fname4 ftype4 val4<44 ) && (fname5 ftype5 val5>5) )
{
```

Similarly, consequent attributes have the same syntax:

```
 9      rule RuleTest1( ( "<%1234-99%>" fname4 val4<44;1 ) && (fname5 ftype5 val5>5) )
10      {
11          [Slots(comments="comment for RuleTest1:Activate cons")]
12          [Active(val=false )]
13          Activate(asdf asdf asdf)
14          [Slots(comments="comment for RuleTest1:quit cons")]
15          quit(true)
16      }
17      [Rule2(val="goalval1" val9="goalval2" )]
18      rule RuleTest2( ( fname4 ftype4 val4<44 ) && (fname5 ftype5 val5>5) )
19      {
```

The attributes currently available for use are the following:

- `[Slots(comments="comment for the goal GoalTest" verb=Change noun=GoalName adverb=Rudely)]`

- `[Activation(returntobase=11 basestrength=22.1 decaystrength=33.1 deletethreshold=44.1 baselinethresholdsigma=55.1 activationthresholdsigma=66.1)]`
- `[Active(val=false )]`
- `[Decays(val=true )]`

A useful ability is to have certain goals not visible to the production system and the hearing system so they will not be executed accidentally (or on purpose) by speaking their names. Another reason would be to reduce the clutter in the production window, allowing only a select series of goals at various times to be displayed. To effect this, the following attribute line should immediately precede the goal declaration line in the .rbt file (as the previous examples do):

- **`[Slots(ShowInEditor=false)]`**

Note that any slot key/value can be added for whatever reason desired by simply following the previous example; they need not be limited to comments, verb, noun, and adverb. Also, the Active attribute controls whether or not a goal, rule, or consequent will be fired when executed. In the goal editor dialog provided by the SS-RICS interface, the indicator that a goal, rule, or consequent has been inactivated is the "#" sign preceding the name (Fig. 48).

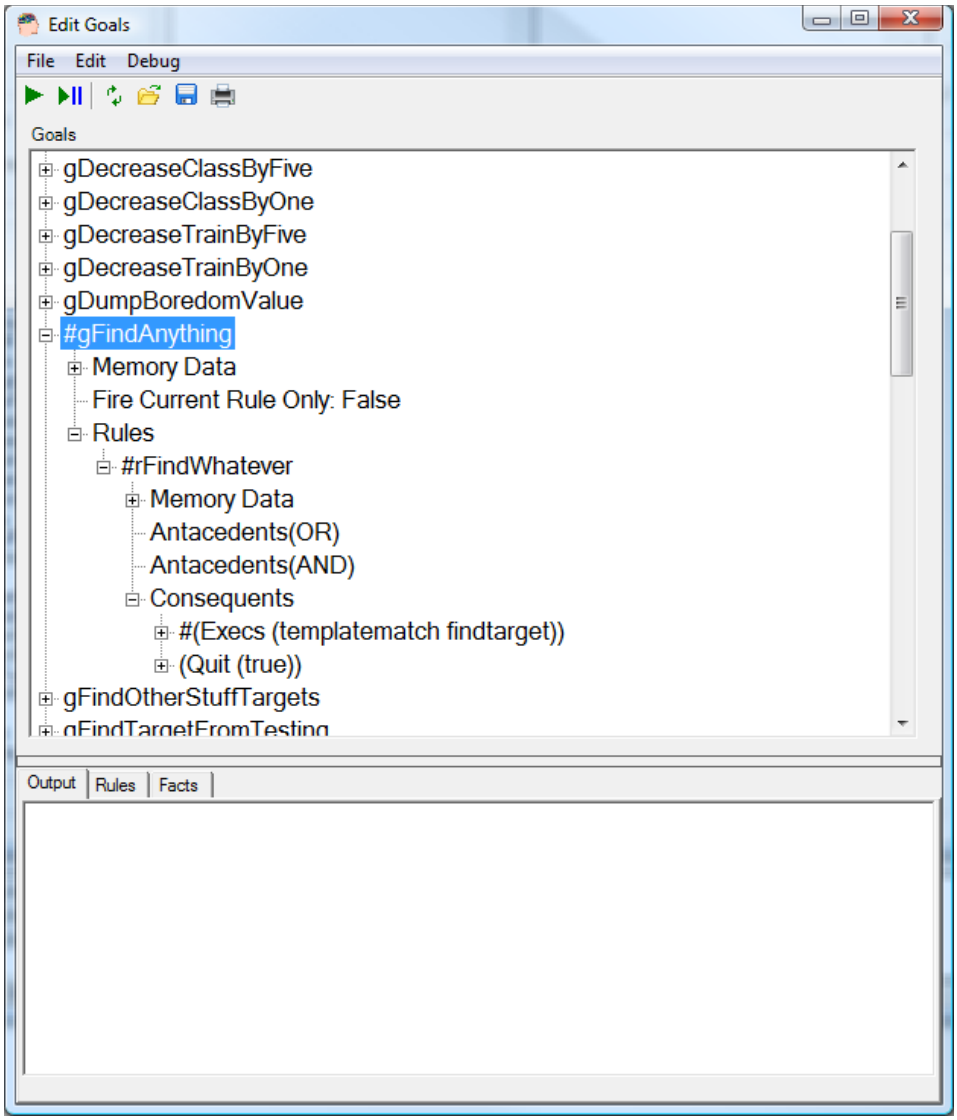


Fig. 48 Goal and rule labeled inactive with "#"

4.5.7 Error Handling

As shown in a previous example, errors are detected in real time as they appear and are noted by a squiggly red line under the offending text and by a detailed explanation in the error box below the code window (Fig. 49).

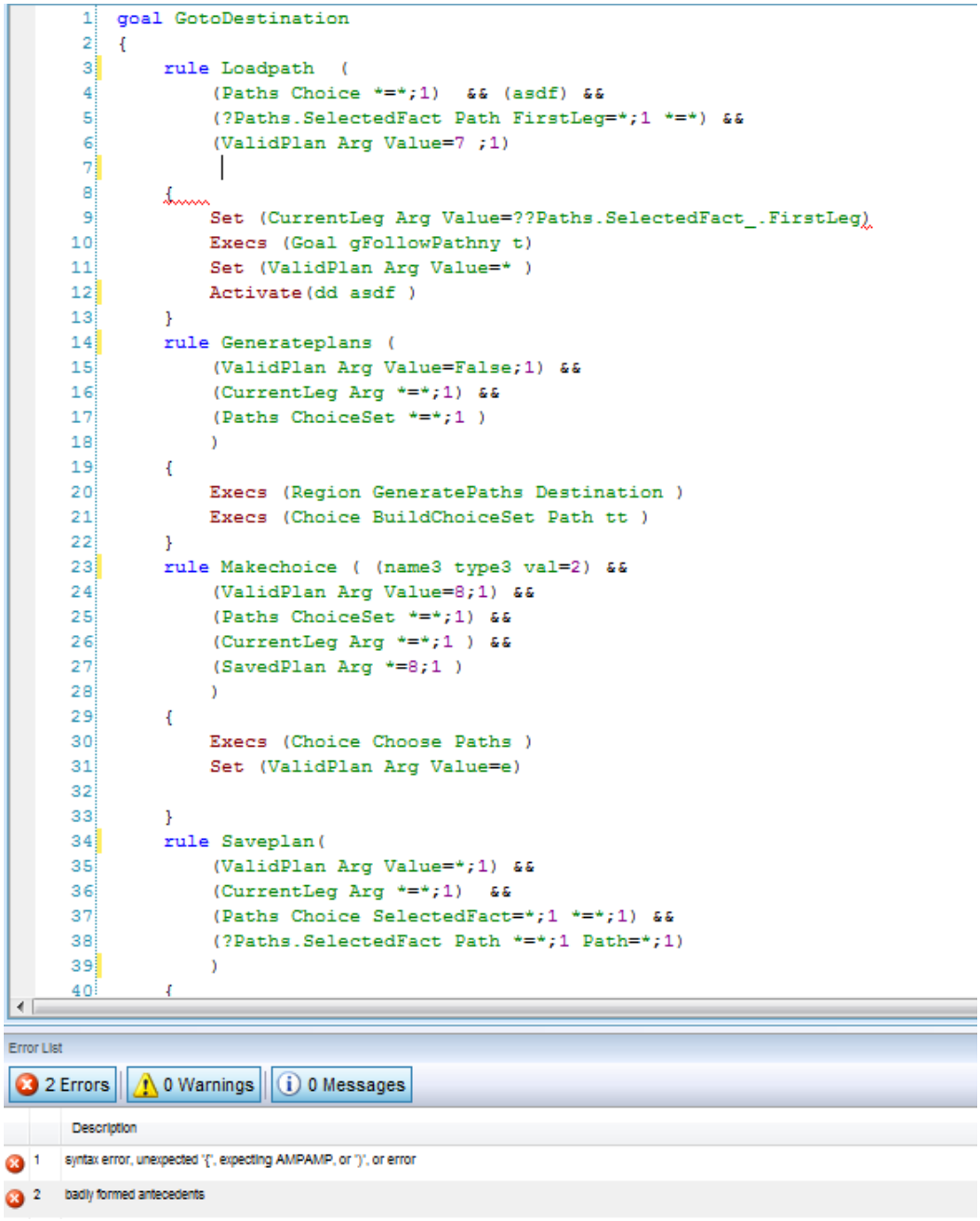


Fig. 49 Errors masking other errors

The red squiggly line is on the "}" following where a ")" should be, and the error list at the bottom reflects that. This is not the only error on the page, however. Because the parser can get lost or overwhelmed when numerous errors are present, it will fail to reveal all of them at once. This happens often in C and C++ compilers and others (notably COBOL [common business-oriented language]). When some of the errors are fixed, the parser can get further and will show errors that followed the one repaired (Fig. 50).

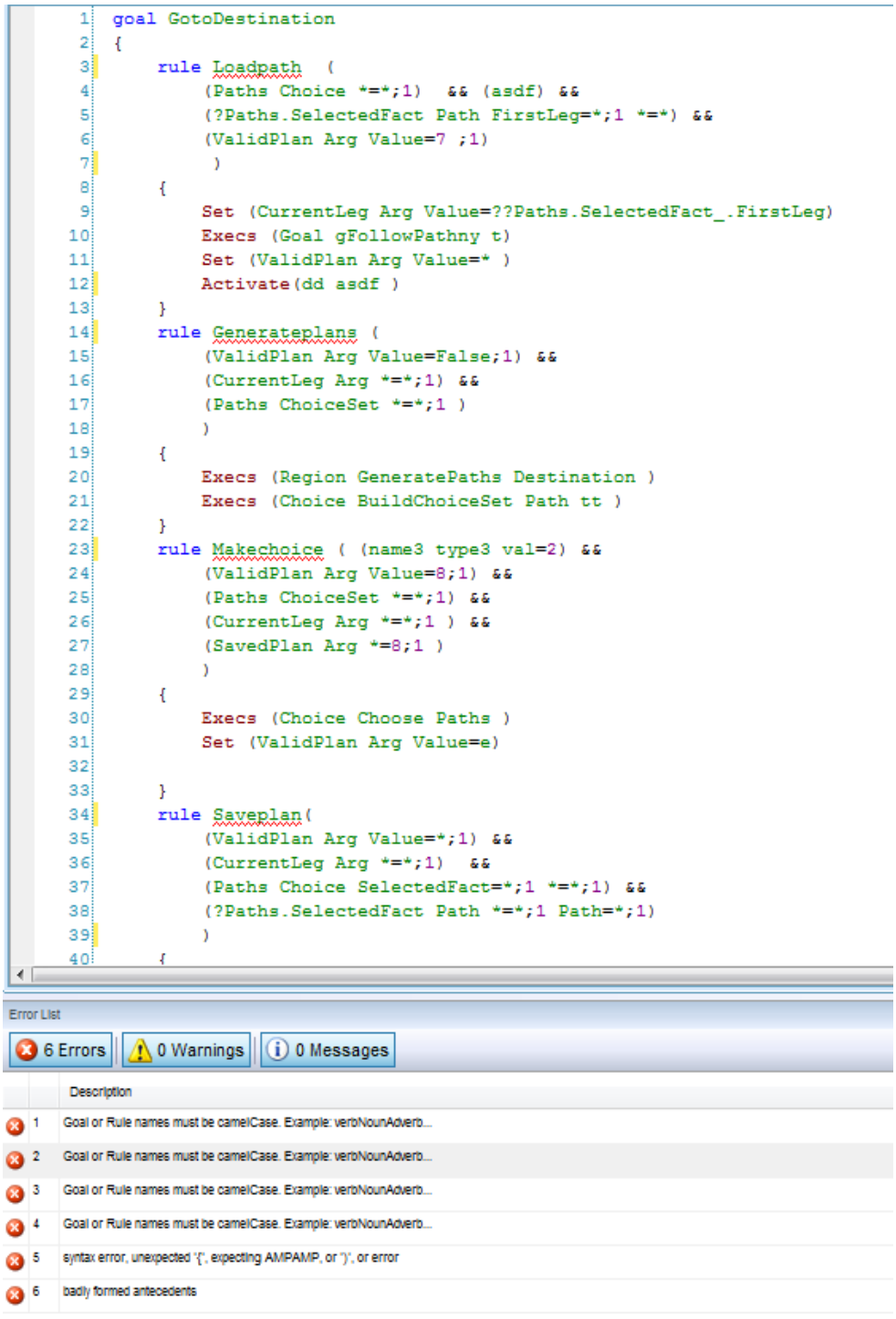


Fig. 50 Masked errors revealed after initial errors corrected

In Fig. 50, the “)” was added where it was supposed to be, and a whole slew of other errors following it appeared. This is normal behavior.

4.5.8 Importing Goals

Pre-existing goal files created with the brain interface goal editing utility and/or goals created and compiled using this facility can be imported into the goal editor by right-clicking on the name of the project (Fig. 51).

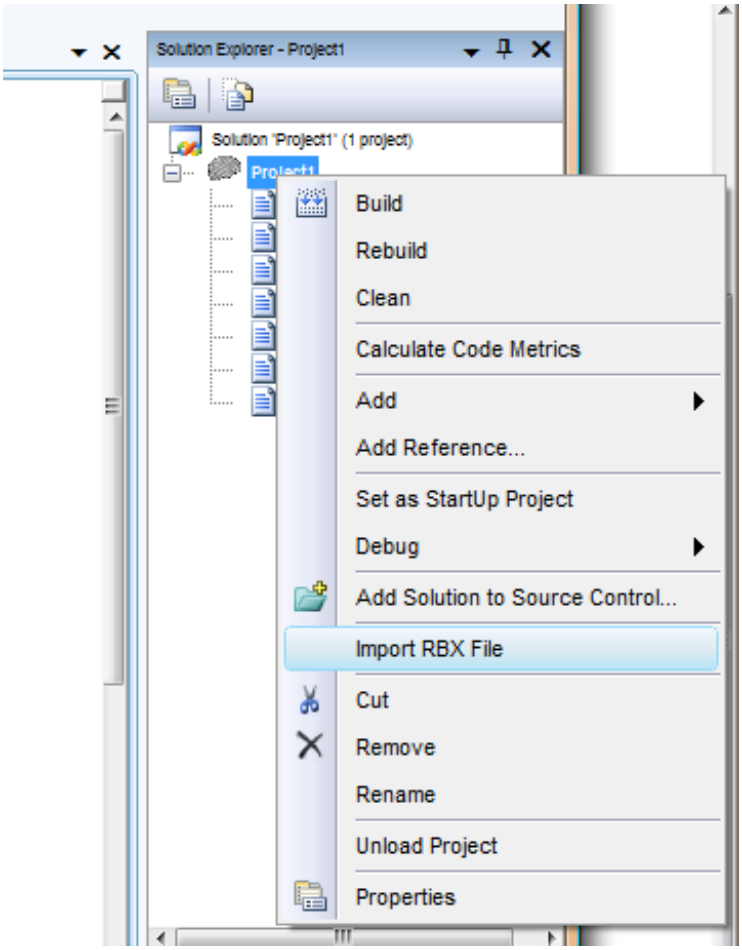


Fig. 51 Goal processing options, import goals file

Then a menu will appear and you will select Import RBX File, which will display a File Open dialog box wherein you will choose the file to open (a screen capture of this menu could not be obtained). The contents of the file will then be displayed in the grammar and syntax of the goal editor, and you may then make any changes you wish.

4.5.9 Exporting Goals

Goals are exported by several means, all of which accomplish the same thing. When the project is ready to be compiled, the user can do any of the following:

- Press F5

- Right-click on the project name in the solution explorer window and select Debug->Start New Instance
- Press the green arrow button on the debug toolbar

The result will be an instance of the brain interface for SS-RICS launched with the goal editor window opened and the goals you created loaded and ready for execution.

A file to be aware of when compiling and executing the project is the configuration file, which allows for running the SS-RICS brain interface. This file by default is called ss-rics.config and is an extensible markup language (XML)-based document. A baseline document is supplied and edited prior to execution of SS-RICS to reflect the name and path of the goals.rbx file being generated. You may customize it to open SS-RICS with whatever subsystems, maps, cameras, and the like. You may wish to be active when SS-RICS starts up.

The following is an example of how to do that as specified in the config file:

```
	<appSettings>
		<add key="hpUsersManual.HelpNamespace" value="(None)" />
		<add key="Map" value="Maps\office.map" />
		<add key="Connection" value="Simulator" />
		<add key="Goals" value=" $(TargetGoals) " />
		<add key="AutoOpenGoalEdit" value="True" />
	</appSettings>
</configuration>
```

Note that the environment variable $(TargetGoals) designates the goals that you just compiled and wish to now test. It is set internally.

For a more detailed explanation of the config file, see Section 14.

4.5.10 Miscellaneous

When using the editor dialog in the SS-RICS interface, the names of goals and rules have a “g” and an “r” prepended to them, as shown in any of the previous screen shots. This is internal to the SS-RICS system, and the names do not appear in the text of a goal in the goal editor as described in this session. If, for instance, you want to create a goal called MyGoal in the goal editor, you would simply type the name as such without a “g” at the beginning of it. When the goal is compiled, the editor adds the “g” or “r” as required, and if you open said goals in the goal editor dialog from the SS-RICS interface, you will see it there.

4.6 Memory Activation Viewer

The Memory Activation viewer (Fig. 52) allows you to view the activation strength of memories in real time as the SS-RICS operates. The viewer is under the View menu, then select Memory Activation. Simply check the memory that you want to view and the graph of that memory's activation will be displayed in real time. Right-clicking on the graph will bring up a menu that will allow you to zoom, print, save image, and copy the image.

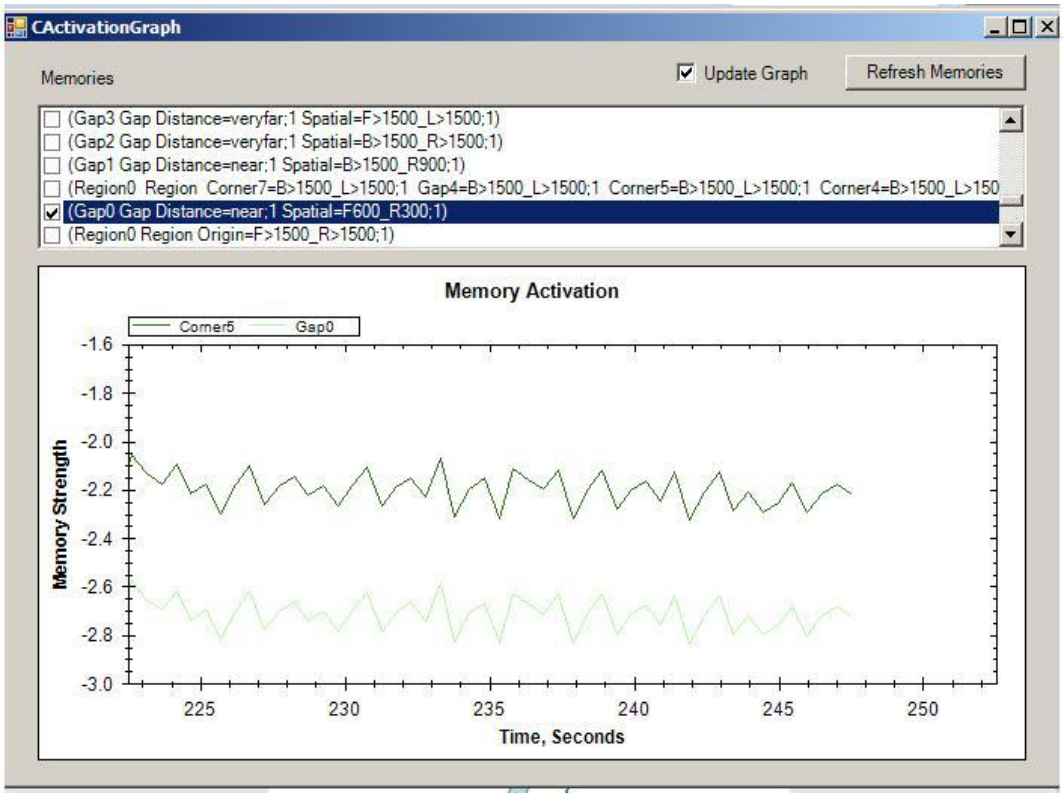


Fig. 52 Memory Activation graph viewer

5. State Memories

SS-RICS creates state memories at start-up that can be used for matching in the production system. To view the memories, choose the Edit menu and select Goals. Then select the Facts tab at the bottom of the window. The memories are of type State and can be selected from the Facts tab. The memories contain information about the current view, the hardware state including motor velocity, battery voltage, laser connection, and position state (theta), and the boredom level of the robot. The RangeRight, RangeLeft, and RangeFront information tell how much space is available on each side of the robot and the front, respectively, using the SICK laser.

6. Production Generation

6.1 Substitution

Substitution learning is the process of creating rules from other rules based on conceptual knowledge. It can auto-generate productions using the Auto Generate Like Goals menu option off of the goals displayed in the goal tree on the goal edit interface, as in the following example: TurnRight can auto-generate TurnLeft.

The distinctions for auto-generation have not be clarified in detail; at this time, SS-RICS will just generate goals with verbs that it can match with verbs. This can create some productions that do not make sense. Further refinements of this type of learning will be made to SS-RICS in future versions, especially as ConceptNet becomes more integrated.

Example:

Initial Rule:

```
rRobotTurnRight
{
        (NothingRight)
=>
        Execs(TurnRight 10))
}
```

Facts:

(Robot Concept IsA= noun) (Turn Concept IsA= verb) (Right Concept IsA= Adverb)
(Left Concept IsA=Adverb)

Created Rule:

```
rRobotTurnLeft
{
        (NothingLeft)
=>
        Execs(TurnLeft 10))
}
```

6.2 Variablization

Variabilization learning is the process of creating rules from semantic relationships of facts or concepts. For the variabilization learning to work, several facts must exist, including the following:

- The Self fact must be created with a slot ID of CapableOf, which defines the actions that the robot can perform.
- The items in the CapableOf slot must also be defined as concepts.

Once these requirements are met, the following process is executed:

1) Find an action the Self is capable of.
2) Find the properties required by the action.
3) Build the rule based on the action concept.

Example:

Facts:
(Self Robot CapableOf=Turn)

(Turn Concept IsA=motion PropertyOf=Direction,Degree)

(TurnDegree Arg Value=10)

(TurnDistance Arg Value=Right)

(Left Concept IsA=Direction,Verb)

Created Rule:

```
rRobotTurnX
{
        ($TurnDirection)
        ($TurnDegree)
=>
        Execs(motion Turn$TurnDirection $TurnDegree))
}
```

The rRobotTurnX rule was created based on the fact that the Self fact is capable of the action Turn. The learning process then creates the rule using the data found in the Turn concept fact. The created rule rRobotTurnX requires the matching of the antecedents $TurnDirection and $TurnDegree, which were generated by the PropertyOf values Direction and Degree in the Turn concept fact. The consequent Execs(Turn$TurnDirection $TurnDegree) was built based on the PropertyOf values, as well as the IsA value, Motion. Thus the rRobotTurnX rule expects 2 parameters, TurnDirection and TurnDegree, which are then passed into the Execs call in the consequent, and the robot is turned the number of degrees supplied in the TurnDegree parameter and in the direction supplied in the TurnDirection parameter.

Merging Facts for Matching

The production system, as well as the lower-level processors, have the ability to collect current facts of a specific type and combine them into a new rule. This new rule will fire anytime the system reaches a state that matches the current memory structure. The consequent of this rule is to add a fact to memory identifying that the previous state has been reached.

Example:

Consequent:
Execs(CreateScene Wall123))

Facts:
(Line0 Line Direction=Right Distance=Far)
(Line1 Line Direction=Left Distance=Far)
(Line2 Line Direction=Front Distance=Close)

Created Rule:
```
rWall123Found
{
        (Line0 Line Direction=Right Distance=Far)
        (Line1 Line Direction=Left Distance=Far)
        (Line2 Line Direction=Front Distance=Close)
=>
        (Set(Wall123 Scene value=true)
}
```

7. Sub-symbolic Processors

The bulk of the user interactive functions that SS-RICS has to offer are performed by individually threaded subprocesses called sub-symbolic processors (or subsims for short). These include subsims that perceive the environment through laser scans and can be used to build planar maps of the environs. Several subsims process images from the camera stream to perform object recognition, segmentation, and tracking. Some can be chained together such that the output of one is the input to another. An example of this is the set of subsims that builds the map of the robot's locale: The Points subsim sends laser point scans to the Linesegment subsim, which sends line segments to the Line subsim, which sends lines to the Intersect subsim, which sends intersections to the Corner subsim, which sends corners to the Gap subsim, which with the Corner subsim sends gaps and corners to the Region subsim. Others send face, motion, and moments detections to the general PredatorTLD tracking/learning/detection (TLD) algorithm tracking subsim. They all share some properties, as described in the next section.

7.1 Common Properties

Many of these subsims produce facts that contain the results of operations performed by the subsims at the behest of the operator/user. These facts have a default type, which is the name of the subsim as specified in the config file. For instance, the TemplateMatching subsim is called templatematch in the config file, so the type would be templatematch. Each subsim provides a command that can be issued to it from the production system (through an Execs command in a goal) that will allow the user to change this value to whatever they wish, as follows:

Execs(<subsimname> setfacttype newtypename), such as

Execs(faces setfacttype facefact) or

Execs(line setfacttype laserscan)

Another property exists for all subsims that is intended for developers: debug level. Three such levels are available: Silent, Normal, and Verbose. They are relatively self-explanatory and used to throttle the amount of diagnostic information displayed in the output console. This value is currently only settable from the property pages of each running subsim.

7.2 Boredom

Three subsims determine whether or not the robot is "bored" or "excited": aural (sound-based), visual (image-frame-based), and laser-scan-based subsims. Each collects data of that type over specified intervals and completes a histogram comparison of each sample to each sample and employs the following algorithm to determine which state it is in.

General Algorithm

To determine novelty, we assess how much variance there is in a sequence of consecutive samplings (equivalent to a time interval) specified by an interval parameter. If the variance does not meet certain criteria, the environment is not considered novel. Each sample is correlated with the others and those values are collected in a histogram matrix. If the correlation value of a particular comparison is greater than a set gradient value, it means that the 2 samples were similar enough to count as the same and thus add to the count of correlation values used to calculate novelty. The value of gradient determines the amount of change required to trigger a potential detection of a novel environment. A low value means that a great deal of change is necessary, and a high value means a relatively small amount of change is necessary.

Algorithmic Flow

Let **α** = vector of observations.

Let **β** = number of observations in the vector **α**.

Let **μ** = matrix of observation correlations (2-D).

Let **T** = threshold value for a correlation value to be considered bored.

Let **B** = percentage of **T** for the robot to be considered bored for the given set of observations.

Let **γ =** number of correlations that exceed the boredom threshold **T**:

```
γ ⟵ 0
for i = 0⟶ β – 1
        for j = 0⟶ β – 1
                if i == j continue
                μ i,j ⟵ correlation(α i , α j)
                if μ i,j > T then γ⟵ γ + 1
        end
end
```

or, where x = correlation(α_i, α_j):

$$\gamma = \sum_{\substack{i,j=0 \\ i \neq j}}^{\beta-1} f(\mathrm{x}) > \mathrm{T}$$

Let **τ** = percentage of correlations that exceed the boredom threshold **T**.

τ = **γ**/ ((**β**^{2} – **β**)/**2**)

```
if  τ > B then
        RobotStatus ⟵ "BORED"
else
        RobotStatus ⟵ "NOT BORED"
end
```

So, if **τ** exceeds the value of **B**, the state of the robot can be said to be bored. This value (as long as this subsim is running) is posted in a fact of type State and name of the subsim that calculated it. In Fig. 53, all 3 are represented: visualboredom, auralboredom, and boredom (names specified in the config file for each subsim).

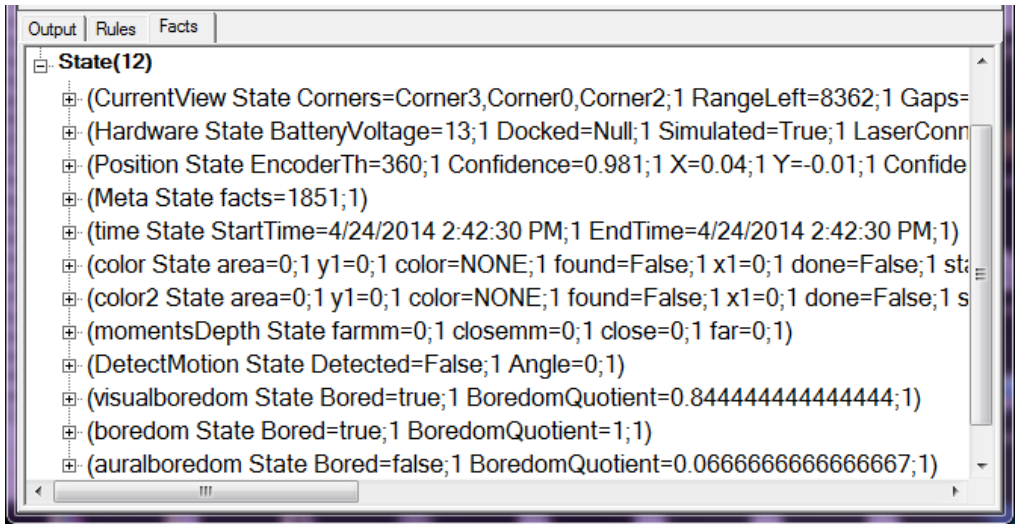


Fig. 53 Fact reporting boredom level

Additional Commands

Execs (<boredom subsim> saveevents <filename>)
Sets the name of the file that will save events as generated with a call to LogEvent. The filename is appended with the extension .csv.

Execs (<boredom subsim> setinterval)
Sets the sampling interval. For (laser) boredom this is the number of consecutive laser scans; for visual boredom this is the number of consecutive frames; and for aural boredom this is the number of seconds.

Execs (<boredom subsim> setgradient)
Sets the gradient value as described; it is the same definition for all boredom subsims.

Execs (<boredom subsim> setthreshold)
Sets the threshold value as described; it is the same definition for all boredom subsims.

Execs (auralboredom startrecording <filename> <duration>)

Sets the name of the file (appended with extension .wav) to which sound is to be recorded; sets the duration of the recording in seconds; and starts recording.

Execs (auralboredom stoprecording)
Stops recording and writes the buffer to disc using the filename specified in the startrecording command.

7.3 Choice

This subsim uses DFT to choose items in a set based on user-defined criteria. Give it a choice set with the items in question (facts that represent the choices) and a set of preferences (also embedded in facts) and it will render a decision and produce a graph like the one in Fig. 54.

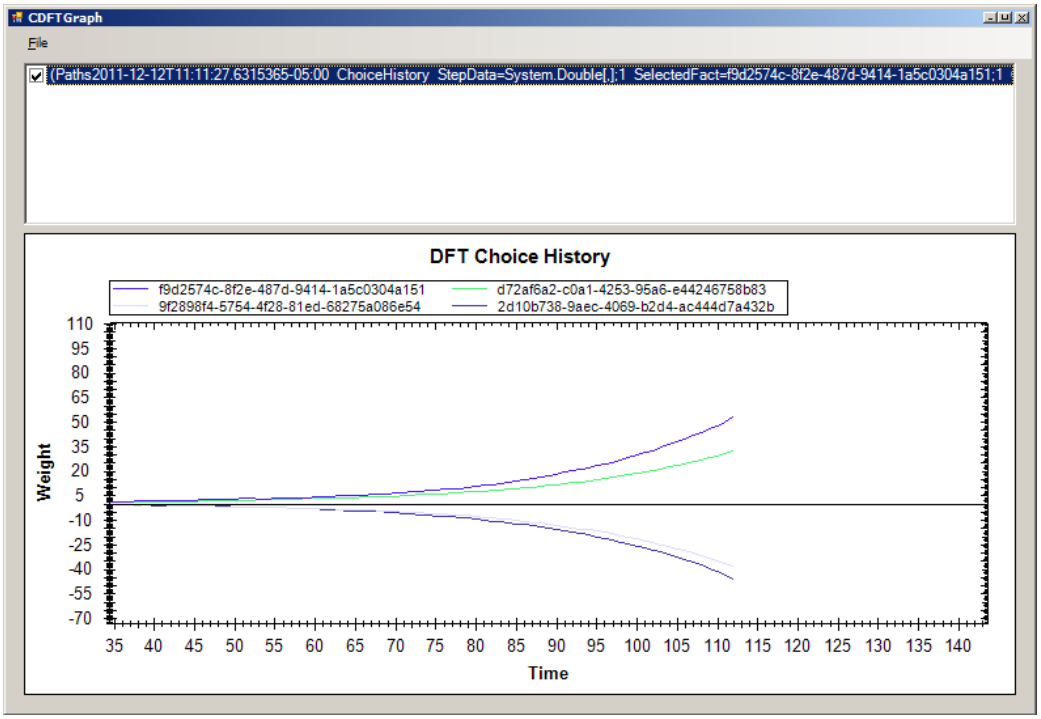


Fig. 54 DFT result graph

One such set of choices could contain paths to choose from, as the graph in Fig. 54 depicts. Danger and length are part of the subset associated with the path choice set, which are features that are weighted (in this case evenly). DFT will process this set given the set of features that will guide it in its calculations. The following is an example command to issue for this function:

Execs (Choice choose choiceFact true)
This tells the Choice subsim to use the choice set contained in the fact named choiceFact and an optional parameter "true" to instruct the system to save the results.

The graph in Fig. 54 shows the final path to decision, and the graph in Fig. 1 shows in more detail the fluctuations that occur in the data as it is being processed.

A set of goals, facts, and map are shipped with SS-RICS and is designed to provide an example to both learn from and build upon. These are found in and/or called the following:

- Goals\Path Follow.rbx
- Facts\PathFacts.txt
- Maps\Hallway – PathPlan.map

7.4 Motion

The Motion subsim handles the movement of the robot and any sensors on it that can perform motion, primarily any pan/tilt/zoom cameras. Commands for this subsim include the following:

Adjustcourse

This is a method for navigating that uses the previous location and current location to chart a course to a new location:

```
goal AdjustCourse
{
	rule AdjustCourse ()
	{
		Execs (motion adjustcourse pX pY dX dY name type delta theta)
	}
}
```

This command computes an angle (to turn the robot to) and a distance to the new location using the previous location specified in the command as well as the known current location and orientation. It then publishes them in a fact whose name and type are specified by the described parameters name and type, with the distance placed in a slot whose key is specified by the parameter delta and the angle placed in a slot whose key is specified by the parameter theta.

7.5 Points

The Points subsim handles collection of laser range readings, assembling them into a series of raster-like sweeps (in this case, 180 readings, 1 per degree) that give a planar view of the robot's environs directly in front of it. These readings can be stored and used to compare with current laser sweeps to determine whether or not the robot is in a familiar place. The Points subsim also sends each collection of laser ranges to the LineSegment subsim so that it can assemble them into candidate lines to use in building a map of the environs.

7.6 LineSegment

The LineSegment subsim receives laser range points and attempts to assemble them into acceptable candidate contiguous line snippets or segments that are sent to the Line subsim for assembly into actual lines that describe the contour of the current area the robot is in.

7.7 Line

The Line subsim receives the line segments from the LineSegment subsim and stitches them together to form lines that will generally be interpreted as walls, depending on length and surrounding structures. These lines are subsequently sent to the Intersect subsim.

7.8 Intersect

The Interset subsim receives lines from the Line subsim and determines whether or not any of them intersect and sends those that do to the Corner subsim.

7.9 Corner

The Corner subsim receives line intersections from the Intersect subsim and identifies them as corners at the junction of 2 walls if they meet certain angle criteria (>60° and <120°).

7.10 Gap

The Gap subsim receives lines from the Line subsim and determines whether or not they are collinear and, if so and not connected, determines whether or not a legitimate gap (most likely a doorway) has been detected. The tolerances for this are settable on the properties page for this subsim. Nominally, these values are 1.2 m (maximum) and 0.8 m (minimum).

7.11 Region

The Region subsim collects corners and gaps from their respective subsims and associates them into consolidated regions provided they are determined to be in reasonable proximity of each other.

7.12 Voice

The Voice subsim listens for voice commands and interprets them as the name of goals. These goals are then executed.

7.13 Haar

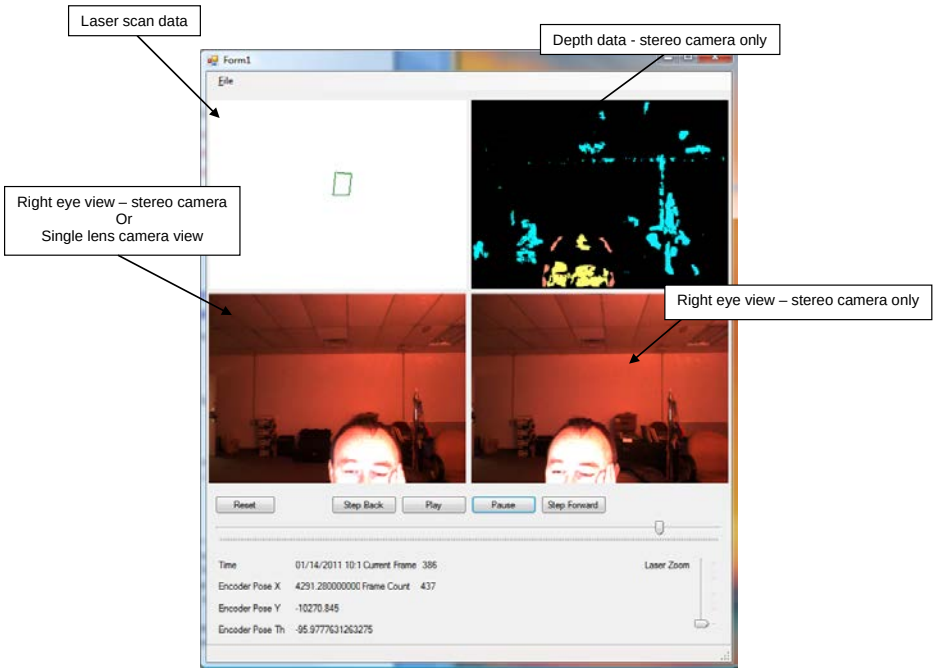
The Haar subsim is a generic shape/object detector similar to the Facial Recognition subsim except that it can employ any Haar cascade available. This includes not only those geared to pick out faces, but also other body profiles including upper/lower and entire. The abilities of the Haar cascades are not limited to the human form, however.

7.14 Data Collection

The DataCollection subsim captures most of the sensor data being generated by the robot and writes them to a file. These data include the following:

- Laser range data
- Robot encoder data (X,Y position, rotation)
- Activity timestamp (the date and time each interval of data was recorded)
- Image data that can include depth data if a stereo camera is in use

These data can be replayed using the DataLogViewer application, the interface for which is shown in Fig. 55.



Note: It is highly recommended the user not start SS-RICS with this subsim activated or, at the least, not activate it with a camera as the collection file can grow quite large quite fast. Use this subsim only when you intend to generate a file and keep an eye on its size. It can reach several gigabytes in just minutes.

Fig. 55 DataLogViewer interface

7.15 Time

Time is a subsim that provides timing data for commands issued in the production system. It allows the production system to determine the time it takes to execute a sequence (one or more) of commands executed in goals. In the following example

```
goal MyCompareScene
{
        rule MyCompareScene()
        {
                Execs (time tic)
                Execs (Line CompareScenes  40 120)
                Execs (time toc)
                Quit(true)
        }
}
```

the Time subsim is invoked with the command “tic”, which tells it to start timing. Upon receiving the command “toc”, it stops and computes the elapsed time between it and tic. It then reports the interval in a State fact (Fig 56).

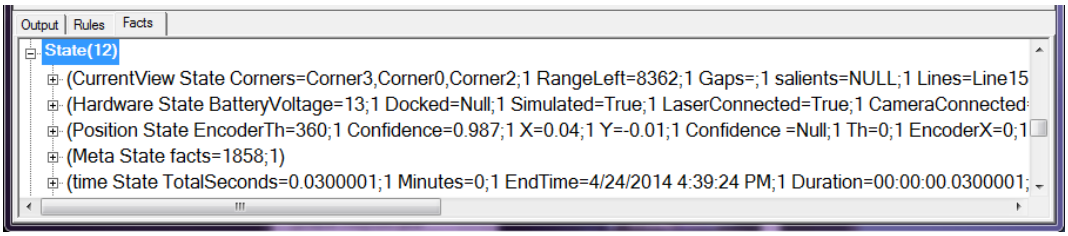


Fig. 56 Report time interval

Additionally, it can create a timestamp of the form MM_DD_YYYY _hh_mm_ss_ms and place it in an Arg fact using the command Execs (time generatetimestamp), as shown in Fig. 57.

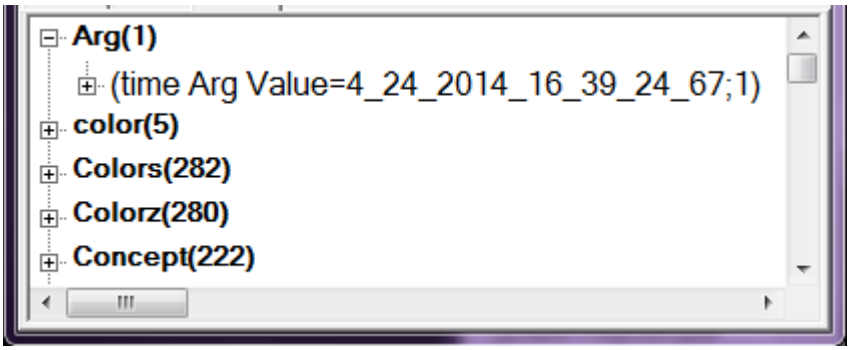


Fig. 57 Generate timestamp

7.16 Laser Correlation

Laser correlation is another way of performing scene designation and classification. It is quite similar to the Image Correlation (discussed in Section 7.17) feature in that it performs the same general list of steps to accomplish the correlation, albeit with different data. Instead of using an image from the camera, Laser Correlation uses a sweep of laser data to create a scene that can later be used for comparisons. Like the ImageCorrelation subsim scene creation and comparison feature, the Laser Correlation feature (hosted in the Line susbsim) is used by creating and executing goals and rules.

To create a scene to use in comparison execute the following command (Step 1):

Execs (Line CreateScene Test Corner Gap)

This goal creates a fact named Test of type RawSceneData containing the following slots: RangeData, which contains the sweep of laser range data; MaxRangeIndex, which contains the index of the highest value in the laser sweep; and MaxRangeValue, which is the highest range value of the laser sweep. The goal, as stated previously, will also create a rule, which will contain all of the current facts in memory of type Corner Gap that describes the current view and location of the robot. This rule can be used later to determine if the robot is in a particular location. If the robot's memory contains at least the same facts as collected in this rule, then it can be said that the robot is in the same location as when the rule was made. This is not used directly in the Laser Correlation function but rather functions as collateral data that can be used anytime after the observation is made.

To compare the current laser scan with the stored RawDataScene created in Step 1, execute the following goal:

```
goal MyCompareScene
{
        rule MyCompareScene((Test? RawSceneData *=*;1))
        {
                Execs (Line CompareScene ?Test 60 120)
                Quit(true)
        }
}
```

This goal will first test to see if a fact of name Test and of type RawDataScene exists, and then it will call upon the Line subsim to execute the CompareScene function against the fact named Test, comparing only the values between 60 and 120 inclusive. This range allows the user to specify a field of view (FOV). The function will also attempt to align the data according to the maximum values in

each as described in Step 1. Once it has accomplished the alignment, it will compare the 2 resulting arrays of data using the Pearson Correlation algorithm and create a fact called CompareScene with a type of Arg and the following slots:

- Value: the rounded-off correlation value between the vectors.
- RawValue: the unrounded correlation value between the vectors.
- SceneT: the T-Value between dependant sample sets (refer to the entry "Students T-Test" in Wikipedia for more information).
- Standard: Student's t-statistic (refer to the entry "Student's T-Statistic" in Wikipedia for more information).
- Significance: the significance of the T-Value (refer to the entry "P-Value" in Wikipedia for more information).

Note that the last 3 slots described are primarily for debug and diagnostic purposes and are not used in the actual correlation reporting.

A variation on Laser Correlation allows the user to execute CreateScene multiple times, producing multiple RawDataScene facts with unique names (specified by the user). An example is when the following goal is executed:

```
goal MyCompareScene
{
        rule MyCompareScene()
        {
                Execs (Line CompareScenes  40 120)
                Quit(true)
        }
}
```

This goal checks the current laser sweep with *each* fact in memory of type RawSceneData and returns a fact named CompareScene with all the same values as previously described but with an additional slot named SceneName containing the name of the RawSceneData that matched the current laser sweep best.

7.16.1 Variant on Laser Correlation

The description of Laser Correlation shares an intentional similarity with the Image Correlation function, which does the same thing except with full camera images. The main difference is that the Image Correlation function allows for saving the created scenes to disk for loading at startup. The Laser Correlation function was created prior to the Image Correlation function and did not have this capability. It

was later modified to allow for the persistence of laser scan data in the same fashion as Image Correlation (see the Image Correlation function description [Section 7.1] for more insight). The use of this function in this mode is almost identical to the original function with the following small exceptions:

Creating Scenes

Create a scene using a goal with the following:

```
goal TestCreateScene
{
        rule TestCreateScene()
        {
                Execs (Line CreateLaserScene Test corner/gap)
        }
}
```

The CreateLaserScene function saves the scene in a directory called LaserScenes in a subdirectory called corner/gap under a file named Test-<date>.data, where <date> is a timestamp to give it a distinct name. (Note that the corner/gap parameter is similar to Corner Gap previously displayed.) This allows/denotes the use of subdirectories beneath subdirectories, in this case LaserScenes/corner/gap/. The contents of this directory would literally be laser scans of corners with gaps. Similarly, the goal to compare the current laser scan with saved scans is almost identical to the original function with the following few small exceptions:

Compare the current laser scan with the stored scenes in the CornerGap subdirectory:

```
goal MyCompareScene
{
        rule MyCompareScene()
        {
                Execs (Line CompareLaserScene corner/gap 60 120)
                Quit(true)
        }
}
```

In this example, the CompareLaserScene function compares the scenes from the corner/gap subdirectory with the current scene. To compare the current scene with *all* subdirectories, the corner/gap parameter is replaced with a “*”, as follows:

```
goal MyCompareScene
{
        rule MyCompareScene()
        {
                Execs (Line CompareLaserScene * 60 120)
```

```
                Quit(true)
            }
        }
```

Comparing a Single Scene

If the Execs line looks like any of the following,

> Execs (Line CompareLaserScene corner/gap 60 120
> 5_15_2012_12_1_21_104) ,

then the laser scene in the fact whose type is corner/gap and whose name is 5_15_2012_12_1_21_104 will be the only one used for comparison.

A variation of the third parameter, the directory to search, could look like the following:

> Execs (Line CompareLaserScene corner/gap/?name.counter_ 60 120
> 5_15_2012_12_1_21_104)

Here, ?name.counter refers to a variable the user creates in a production (see Section 4.3 on the production system and its related syntax), and it will be appended to the directory corner/gap such that if it equals 1 it will resolve to corner/gap/1. This is useful if you wish to create subdirectory names of an incremental nature. Also note that there is a "_" at the end of the directory name. When using variables in a string it is usually wise to end it with a "_". This aids the goal interpreter in parsing the command. More examples follow:

For a specific directory and a specific filename:

> Execs(Line comparelaserscene corner/gap 60 120
> 5_15_2012_12_1_21_104)

For any directory and a specific filename:

> Execs(line comparelaserscene * 60 120 5_15_2012_12_1_21_104)

Specifying a Result Fact

The default fact type for publishing the result of a Laser Comparison operation is Arg. If you wish to specify another type name (since most of the subsims typically report their results in facts of type Arg, this is advisable), use the following syntax:

> Execs (Line CompareLaserScene corner/gap 60 120
> 5_15_2012_12_1_21_104 MyResultfactType)

Then the result fact will be of type MyResultFactType and named CompareScene. Note that this is a positional parameter. It must be the seventh item in the Execs command, as just shown. If you do not wish to specify a particular scene to compare with but instead want multiple scenes to be considered, use the string NULL instead of a particular scene name, as follows:

```
Execs (Line CompareLaserScene corner/gap 60 120 NULL
MyResultfactType)
```

Deleting a LaserScene

LaserScenes and their related facts can be deleted using the following command:

```
Execs (Line deletescene 5_15_2012_12_1_21_104 corner/gap)
```

Then the scene whose fact has name 5_15_2012_12_1_21_104 and type Corner/Gap will be deleted along with the fact. If the name is replaced with a "*", all files in that directory will be deleted along with their associated facts.

7.17 Image Correlation

Image Correlation is typically performed through the use of goals or, as they are also called, productions, in the following process:

- Images are captured, labeled, and categorized. They are stored in subfolders of a folder called Images. Each subdirectory is named for the general category, such as Doors, Offices, or Hallways. Each image in these folders is named for the specific instance of the category. An example is a category, Doors, wherein there is an image called TroysDoor or EricsDoor. Image categories can be further delineated by using subdirectories of the categories such that instead of labeling an image TroysDoor or EricsDoor, you could have a subdirectory for each under Doors: Images/Doors/Troy for images of Troy's door and Images/Doors/Eric for images of Eric's door. There is no set limit (all machine- and space-dependent) to the depth of the subdirectories. One could have Images/Doors/Eric/Home/Back/knobs to contain images of Eric's home's backdoor knobs. This directory structure and storage strategy exists for Laser Correlation, Image Correlation, and Template Matching.
- At any time, the robot can be asked to identify the scene it is looking at. It can then be directed to compare the current image (what the camera sees now) with the images in the subdirectories of the Images folder and render a statistical match using the image correlation algorithm gleaned from MATLAB. This value is reported in a fact whose type is Arg and whose

name is either specified by the user or defaults to ComparedImage. The correlation value is returned in the fact as well as the name of the image that provided the closest match, an example of which is showon in Fig. 58.

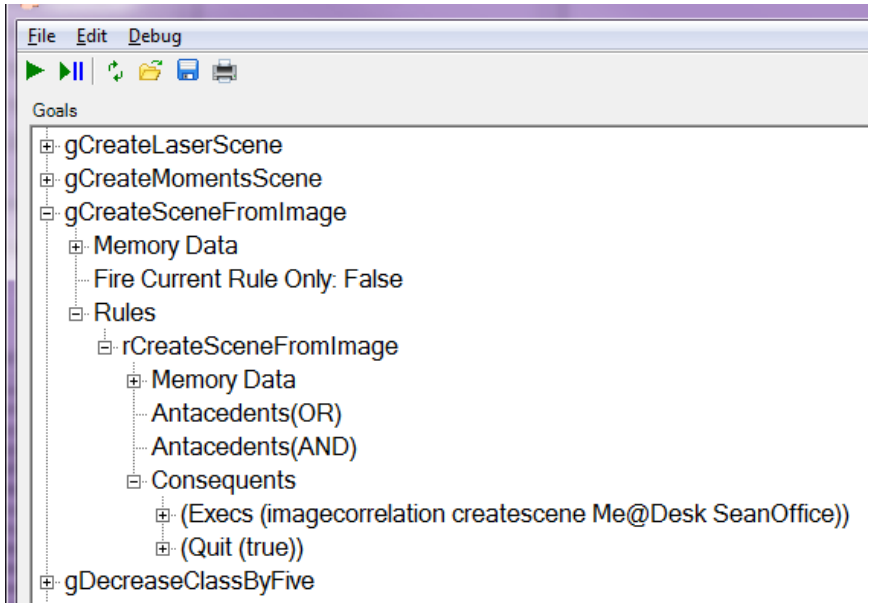


Fig. 58 Goal to create scene for image correlation

This shows the goal that captured the images used for the comparison. They are all images of the interior of the office of someone named Me. Thumbnails of these images are shown in Fig. 59 along with their names and the folder they are in.

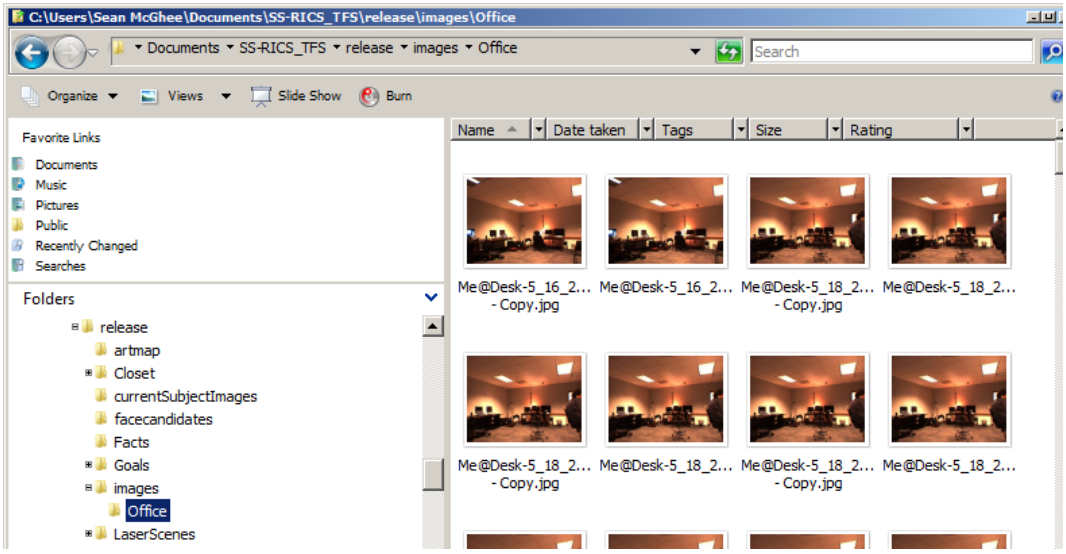


Fig. 59 Captured scene thumbnails

Figure 60 illustrates the image to be compared with the thumbnails.

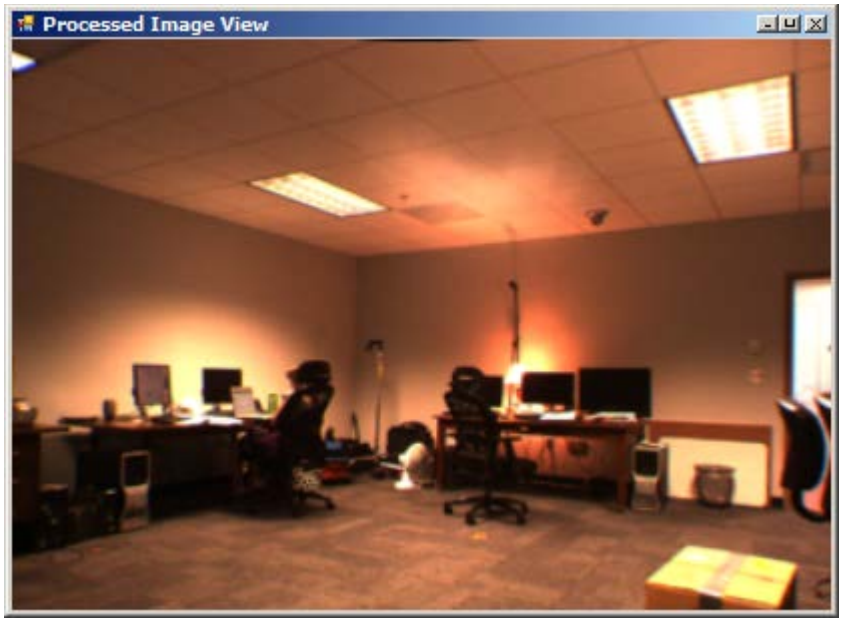


Fig. 60 Image to match using Image Correlation

Figure 61 shows the goal to run a correlation match.

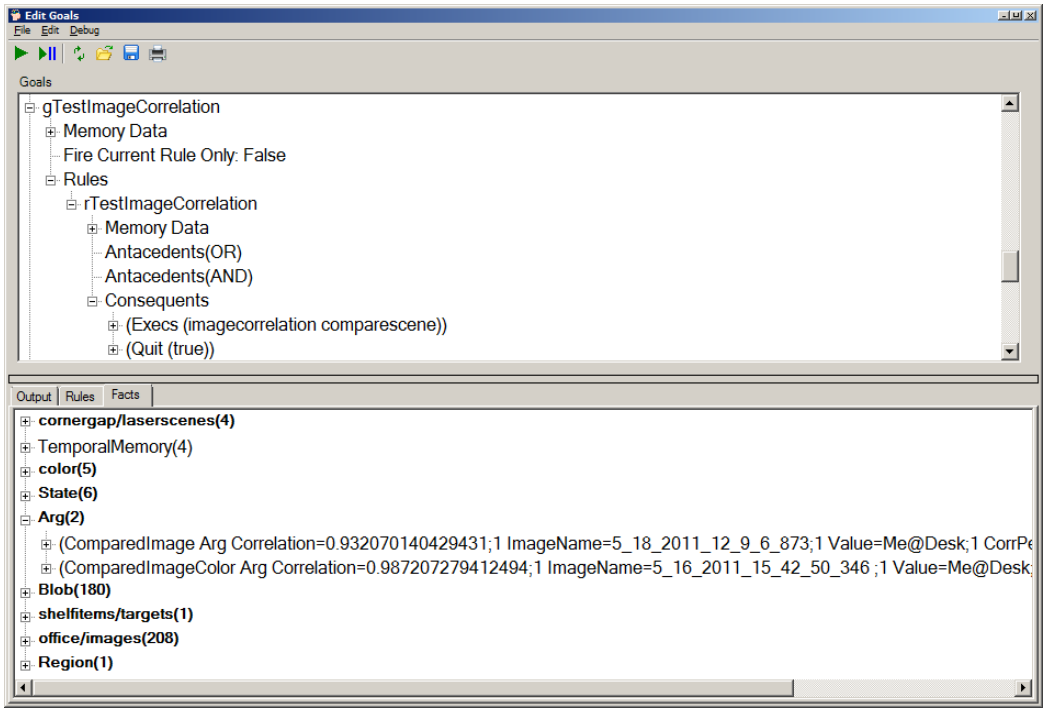


Fig. 61 Goal to run for correlation match (top), with (bottom) results

As seen in Fig. 61, there is a fact in the bottom window under the type Arg whose name is ComparedImage showing a correlation value of .932070140429431 for an image whose name is Me@Desk-5_18_2011_12_9_6_873.jpg.

The goal gTestImageCorrelation as written will search all subfolders of Images (there was only one in this example) and return the result in the generically named ComparedImage fact. A factname can be specified by specifying the following:

Execs (ImageCorrelation comparescene myFactName)

This returns the answer in a fact named myFactName.

Also, you can specify which subfolder to search instead of having them all searched by specifying the following:

Execs (ImageCorrelation comparescene myFactName Office)

Note: To specify a subfolder, you must also specify a factname.

The parameters in this command are the following:

- Execs: tells the system to execute the following command synchronously (in the same thread and blocking some activity)
- ImageCorrelation: denotes the subsim responsible for carrying out the command
- Comparescene: the name of the command to carry out
- myFactName: name of the fact returning the results (if not present and if no subdirectories are specified, factname defaults to ComparedImage)
- Office: name of the subfolder or subcategory to search in (if not present, all subfolders are searched)

If you wish to search all directories in a subfolder, use the following:

Execs (ImageCorrelation comparescene myFactName Office/)*

The "/*" will tell the subsim to look through all the folders directly underneath Office.

Alternatively, if you want to compare the current scene with a particular scene, you can use the following form:

Execs (Imagecorrelation comparescene myFactName OfficE 5_18_2011_12_9_6_873))

The parameter at the end, 5_18_2011_12_9_6_873, is the name of a particular fact whose type is Office and the image specified in that fact will be the only one used in comparison.

Figure 62 shows an example of the output for an image correlation operation.

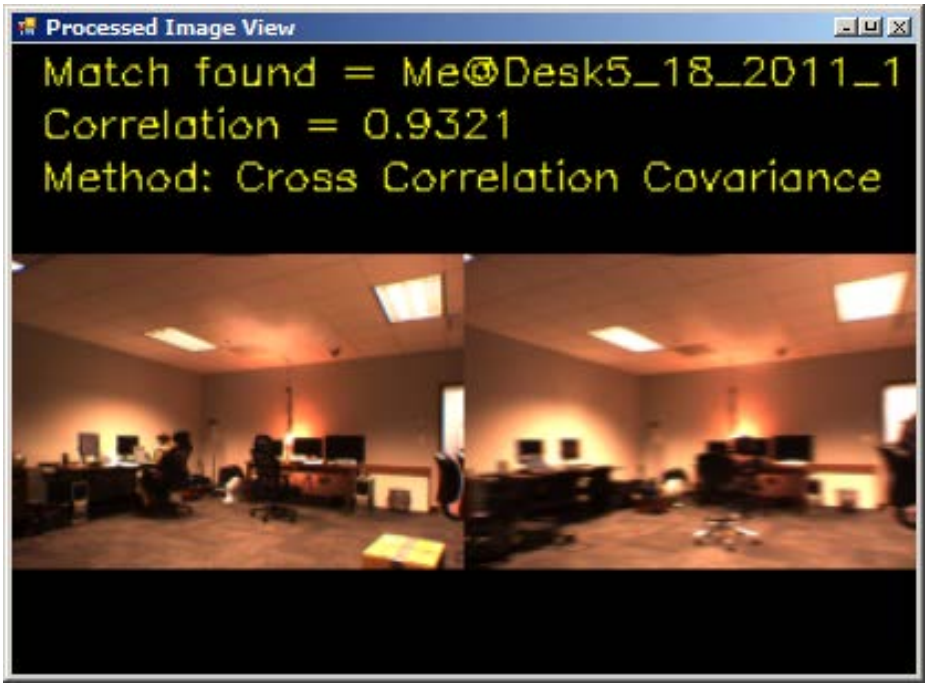


Fig. 62 Cross-correlation comparison result

The image shows (top) the name of the file, the correlation value (highest found in the sampling), and the method of correlation used. Below this information is the current scene (left) and the closest match the robot found (right).

7.17.1 Variant on Image Correlation

The correlation function uses a cross-correlation covariance algorithm to determine the similarity between the images. An alternate method, using color histogram correlation, is available with this subsim. It can be accessed in exactly the same way as described except the command to use is comparecolorscene as opposed to comparescene. Figure 63 is an example of the output from the color histogram method.

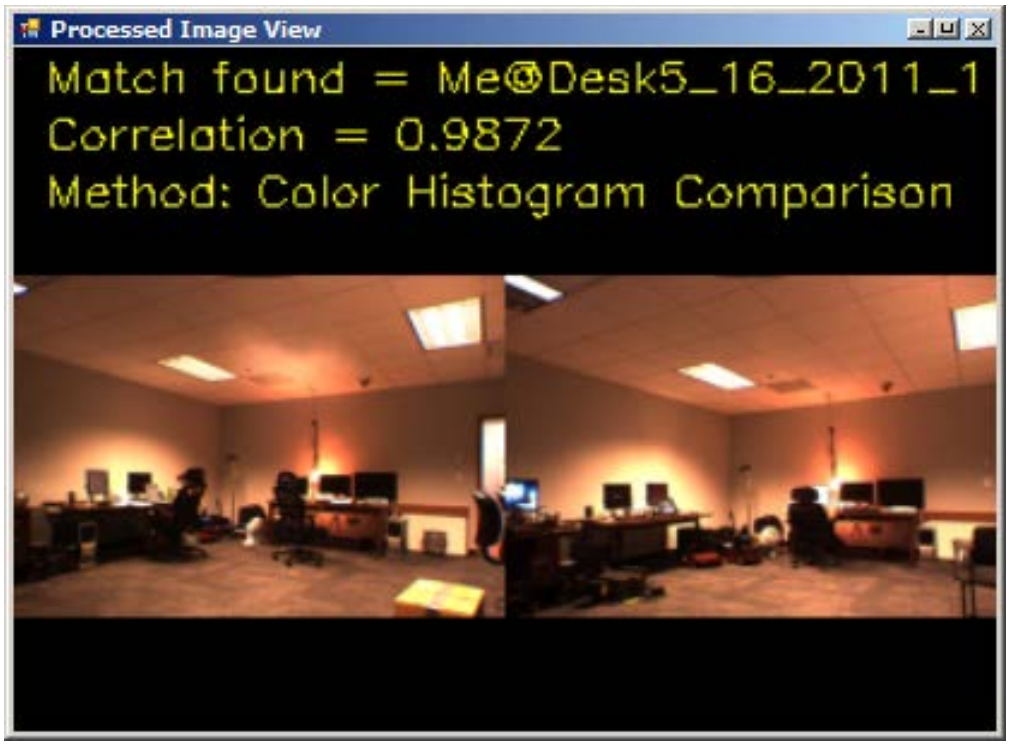


Fig. 63 Color histogram comparison results

As you can see, this correlation appears to be better since we can see that the images are almost the same and the correlation is higher than the previous image, which used the cross-correlation covariance approach. This does not make it more accurate because the 2 methods use different characteristics for comparison. Also, the result fact shown in Fig. 63 in the State Facts collection is identical to the one created for the cross-correlation covariance method except its name is ComparedImageColor instead of ComparedImage.

Additional Commands

Execs (ImageCorrelation quickfindon)
Issuing this command will result in the first match to reach the specified correlation threshold being chosen as a match, which makes the correlation run faster.

Execs (ImageCorrelation quickfindoff)
This command will resume normal searching, and the target with the highest correlation value is chosen. This could mean all scenes are examined before a match is made, so the processing could take several seconds.

Execs (ImageCorrelation setcorthreshold .99)
This sets the value a correlation must reach before being considered valid. If no files are found at the threshold, the entire directory will be searched, and the

next-highest value is returned. The default value is –1, which indicates that only the highest correlation will be accepted. Using this in conjunction with the quickfind option sacrifices potentially better candidates but can result in a much quicker match. This is sometimes desirable when the number of scenes is large. This value is used for both regular correlation and color histogram correlation.

Execs (ImageCorrelation deletescene 5_18_2011_12_9_6_873 Office/images)

*Execs (ImageCorrelation deletescene * Office/images)*

Issuing this command will result in the deletion of the image contained in the fact whose name is 5_18_2011_12_9_6_873 and whose type is Office/Images. Replacing the name with a "*" will result in *all* images in that directory being deleted along with their corresponding facts. Whenever a file is deleted, a cleanup mechanism is engaged to determine if the directory containing the scene has become empty. It recourses through the parent tree and finds and deletes all directories that are completely empty (no files or subdirectories).

Image Facts

Upon subsim initialization, all images subdirectories (images/, etc.) are scanned and a fact describing each is created and displayed in the facts tab of the Goal Edit dialog box. Figure 64 shows some examples:

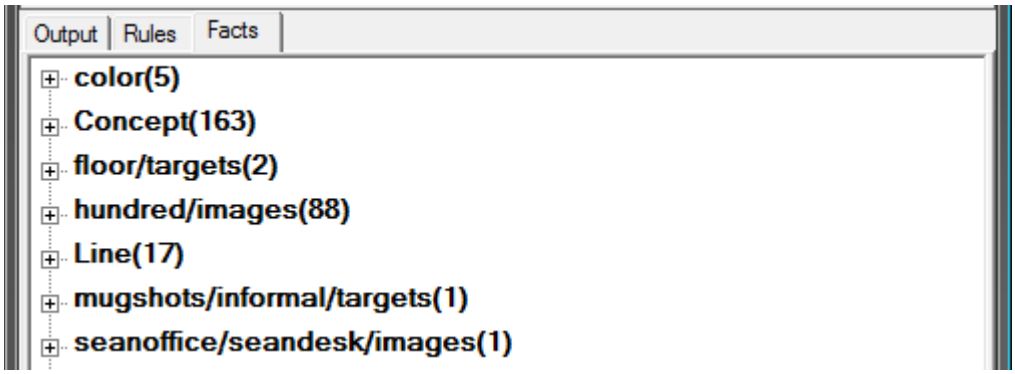


Fig. 64 Image facts in the Facts window

The type of the fact is the name of the subdirectory with the top-level directory appended to the rest of the subdir name (e.g., the subdir name is images/seanoffice/seandesk, so the type is seanoffice/seandesk/images). The fact itself contains the following fields:

- Name: this is part of the name of the file of the image; the timestamp portion as mentioned previously.
- Type: as described previously.

- Year, month, day, hours, minutes, seconds, and milliseconds in individual slots (same as the name only separated).
- Epoch: DateTime object string format of the name; this is the time that the image was saved.
- Label: this is the first part of the file name (e.g., file name is me@desk-7_8_2011_10_23_0_500; the label is me@desk).
- ImageName: this is the complete full path name of the file starting with the drive letter.
- TheImage: if image data are preloaded, this slot contains the image matrix.

Image Correlation over a large number of images can result in an out-of-memory exception. This happens if the images are loaded during startup. Accessing them from memory incurs memory usage and can cause an out-of-memory exception. If planning to maintain an extensive cache of images (several hundred/thousands), you might opt to not preload the images. This will incur some input/output (I/O) costs, as the images will all have to be read from disk first, but these have been minimal. See Section 7.1 for information on how to specify this and set it as a default.

7.18 Template Matching

Template Matching is almost identical in use and execution to Image Correlation except that the stored images are snippets from full camera images as delineated by a rectangle drawn on the screen by a user. The multiple subdirectory structure used by the Laser Correlation (the variant) and Image Correlation functions is present in Template Matching as well. To specify one in the creation or comparison of templates, use the same syntax as for Laser Correlation and Image Correlation, wherein the directories are simply denoted by their names separated by a forward slash "/".

The idea is to capture an image of an item, say a detached head (Fig. 65).

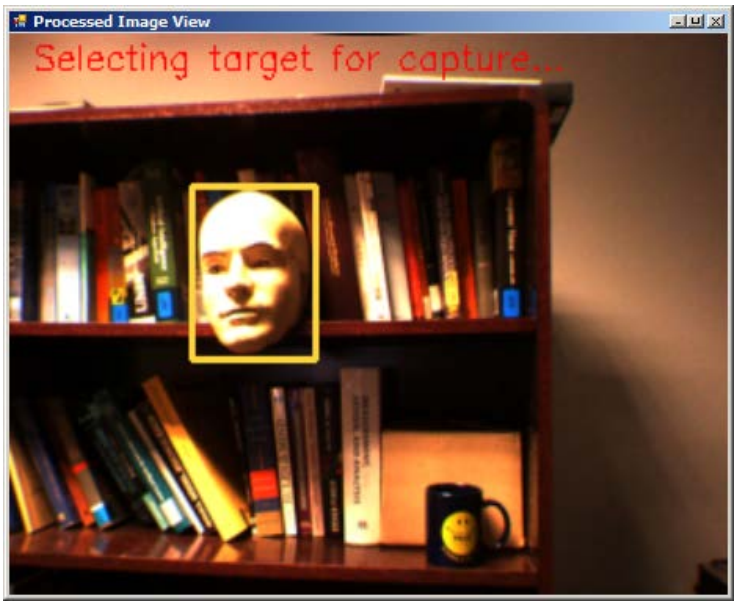


Fig. 65 Capture an image for Template Matching

This image was captured using the goal shown in Fig. 66.

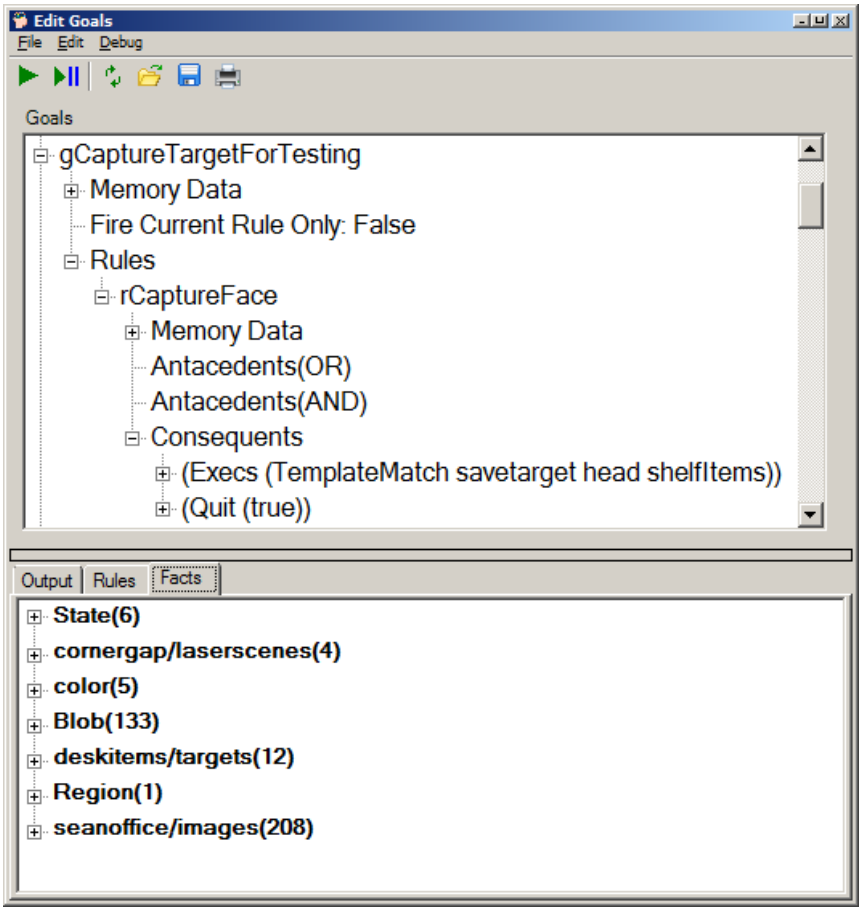


Fig. 66 Goal used to capture image for Template Matching

Execs (templatematch savetarget head shelfitems)

Issuing this command will result in the outlined target being saved to a jpg file possibly named head-6_2_2011_14_14_41_581.jpg (the timestamp portion of the name will vary) in a subdirectory of the targets directory called shelfitems. Figure 67 shows the thumbnail and folder name this image is stored in.

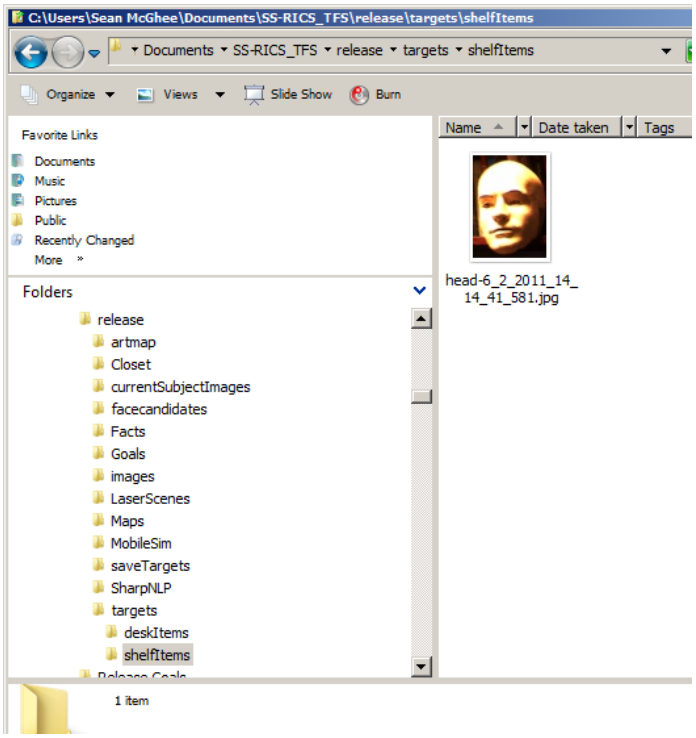


Fig. 67 Captured thumbnail in target subdirectory "shelfItems"

Figure 68 shows the image to be scanned.

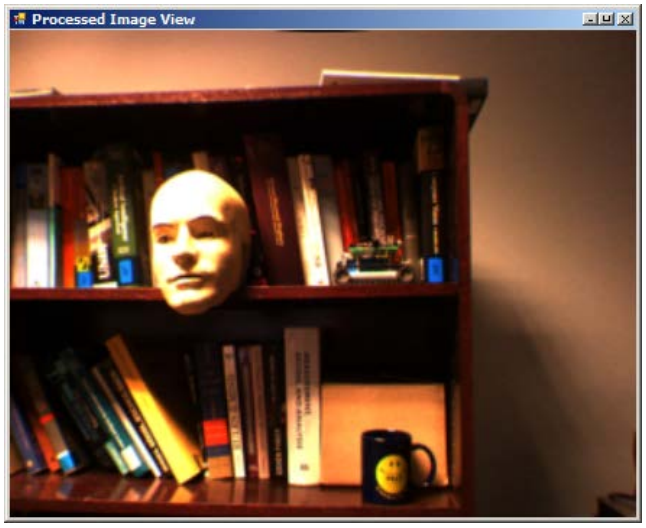


Fig. 68 Image to scan for target

Figure 69 shows the goal used to find the target.

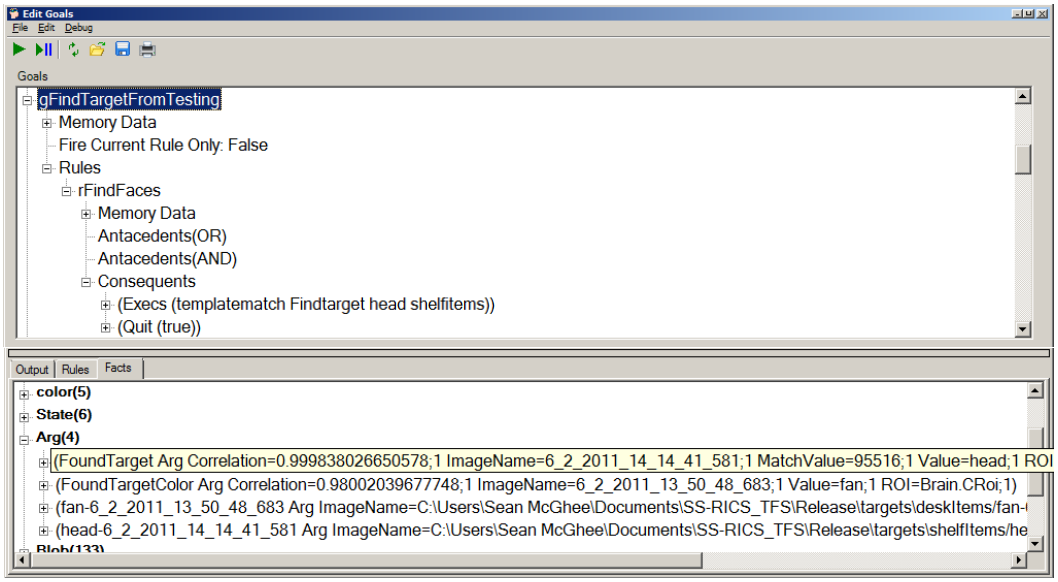


Fig. 69 Goal to find target (top) and results (bottom)

In Fig. 69 the FoundTarget fact (type Arg) has a correlation value of .999838026650578 and has been identified as the image head, 6_2_2011_14_14_41_581.jpg, that you can see in the thumbnail in Fig. 69. The Value slot has a value of "head" that prefixed to the ImageName slot makes up the complete name of the file.

If the target had depth data, the distance to the target from the camera when it was captured and the distance of the target found are also reported in the facts (both FoundTarget and FoundColorTarget) in 2 slots: TDepth (depth of target when captured) and Depth (depth of current target identified). Otherwise, these values appear as NULL. If they are not NULL, an additional slot, DiffDepth, will show the difference between the TDepth and Depth.

Figure 70 shows the results.

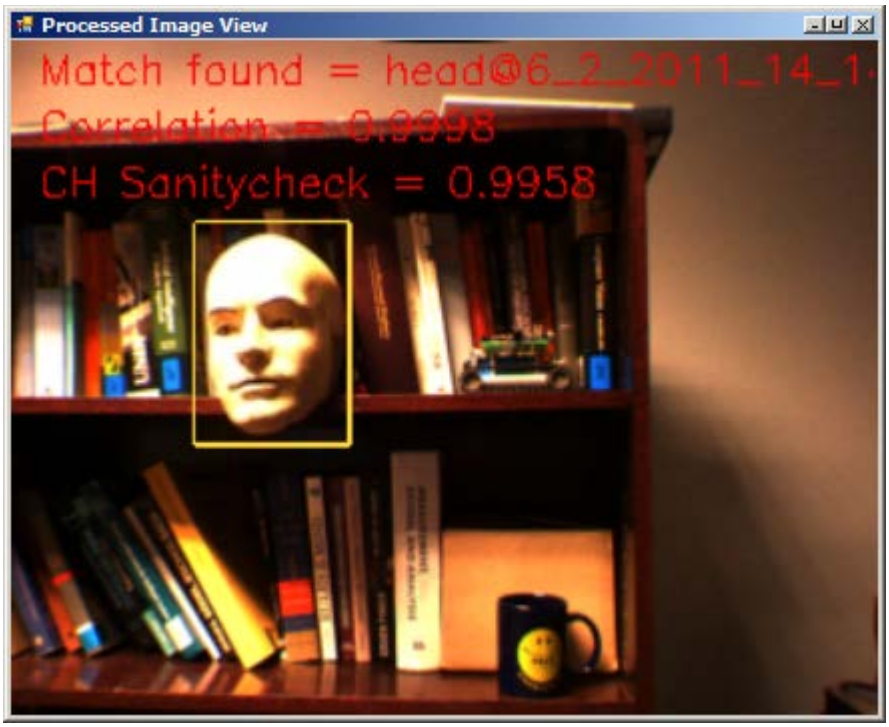


Fig. 70 Target found with target superimposed

Because the image is highlighted with a yellow box and within is a superimposition of the target saved, the viewer sees immediately whether or not the target has been correctly found. Also apparent is the name of the file displayed, followed by the correlation value and another value: CH Sanitycheck with a value of 0.9958. This checks on the result using a color histogram comparison on the image found and the image sought. This is done because the target might not be in the scene, but the algorithm will attempt to find it anyway and come up with the best result even if it is abysmally wrong, as shown in Fig. 71.

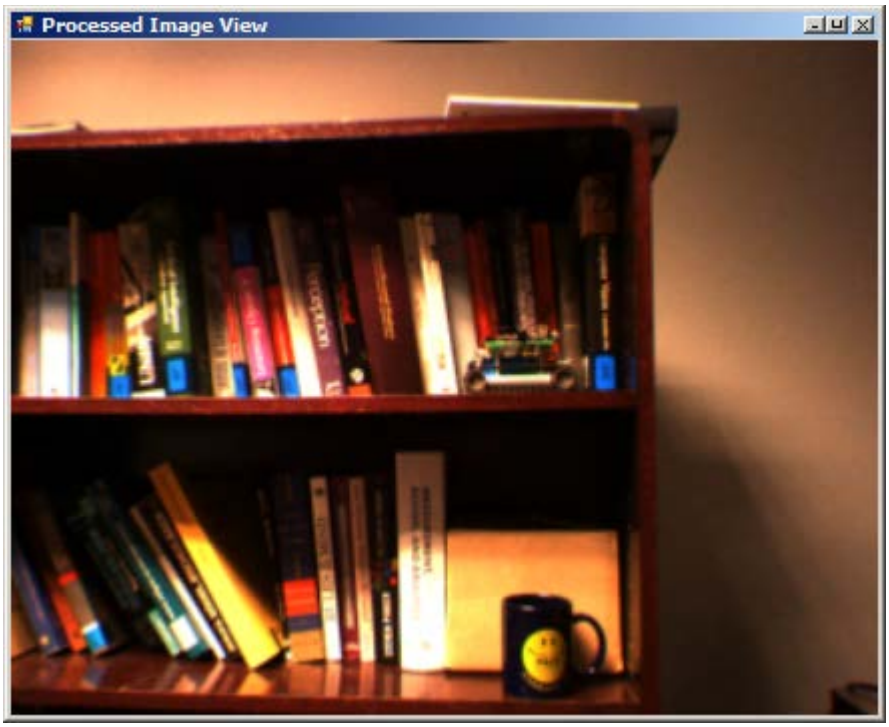


Fig. 71 Image to search for target that is not there

Figure 72 shows the result of running the following goal.

Execs (templatematch findtarget head shelfitems)

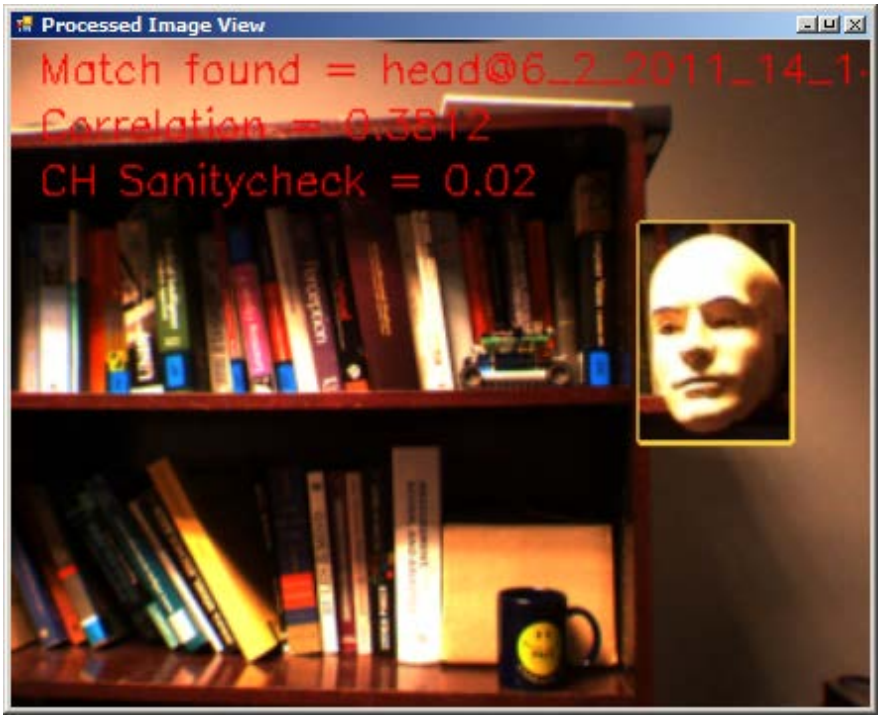


Fig. 72 Egregiously wrong match of target that is not there

This goal specified searching for any item called “head” in the subdirectory shelfitems (as previously illustrated) and claimed that it found the target floating in the air next to the shelves. Clearly this is incorrect, and the values of the correlation (0.3812) and sanity check (0.02) show that. It is also possible for the subsim to find a high correlation value on something that is clearly not the target, but the sanity check will usually show a very low value to adjust the confidence level.

The consequent commands available to the Template Matching subsim follow.

Saving a Target

Execs (TemplateMatch savetarget name directory)
This command saves the target highlighted with the bounding box shown in Fig. 73. The target name is prefixed to the current timestamp and saved as a jpg image in the folder named directory. If the stereo camera is used and the distance from the camera of the target is acquired, it will be appended to the timestamp in millimeters (i.e., 10_3_2011_15_7_25_704**_3749**). If no depth was computed or the camera is not a stereo camera, the characters _NULL are appended instead (i.e., 10_14_2011_14_28_47_555**_NULL**). The facts created for each target captured include a separate slot called Depth with the same information, a distance in millimeters or NULL.

The following example was used in Fig. 73:

Execs (Templatematch savetarget head shelfitems)

The name is head and the folder is saved in the folder targets/shelfitems.

If you want to save in a subdirectory of shelfitems named heads, for instance, the command would look like the following:

Execs (TemplateMatch savetarget head shelfitems/heads)

Finding a target

Execs (TemplateMatch findtarget)
This returns the answer in a fact named myFactName. Since no fact name is specified, the result is returned in a fact of type Arg named FoundTarget. When the target is found, the fact created to the target is returned of type Arg.

Execs (TemplateMatch findtarget myFactName)
This returns the answer in a fact named myFactName.

Since no subdirectory was specified in either of these cases, it will search all of them and find the best match.

Alternately, you can specify which subfolder to search instead of having them all searched by specifying the following:

Execs (TemplateMatch findtarget myFactName shelfitems 5_18_2011_12_9_6_873)

To specify a subfolder you must also specify a factname.

The parameters in this command are the following:

- Execs tells the system to execute the following command synchronously (in the same thread and blocking some activity).
- TemplateMatch: denotes the subsim responsible for carrying out the command.
- Findtarget: the name of the command to carry out.
- myFactName: name of the fact returning the results (if not present, factname defaults to FoundTarget).
- shelfitems: name of the subfolder or subcategory to search in (if not present, all subfolders are searched).
- 5_18_2011_12_9_6_873: name of a specific fact to use in searching. The target image contained in this fact will be the only one used for searching the current scene. If this parameter is not present, the entire directory specified will be searched and the appropriate candidate returned in the result fact myFactName

Deleting a Target

Execs (TemplateMatch deletetarget 5_18_2011_12_9_6_873 shelfitems)

Issuing this command will result in the deletion of the target contained in the fact whose name is 5_18_2011_12_9_6_873 and whose type is shelfitems. Replacing the name with a "*" will result in *all* targets in that directory being deleted along with their corresponding facts.

7.18.1 Variant on Template Matching

The other way to perform template matching is with color histogram comparisons alone. The syntax for commands and usage are identical to the method described except that the command to find a target is findtargetCH. This method is *extremely* slow due to a lack of optimization code for it, so its usage is not encouraged for casual identification. Figure 73 shows the result of a command to find a target using this method.

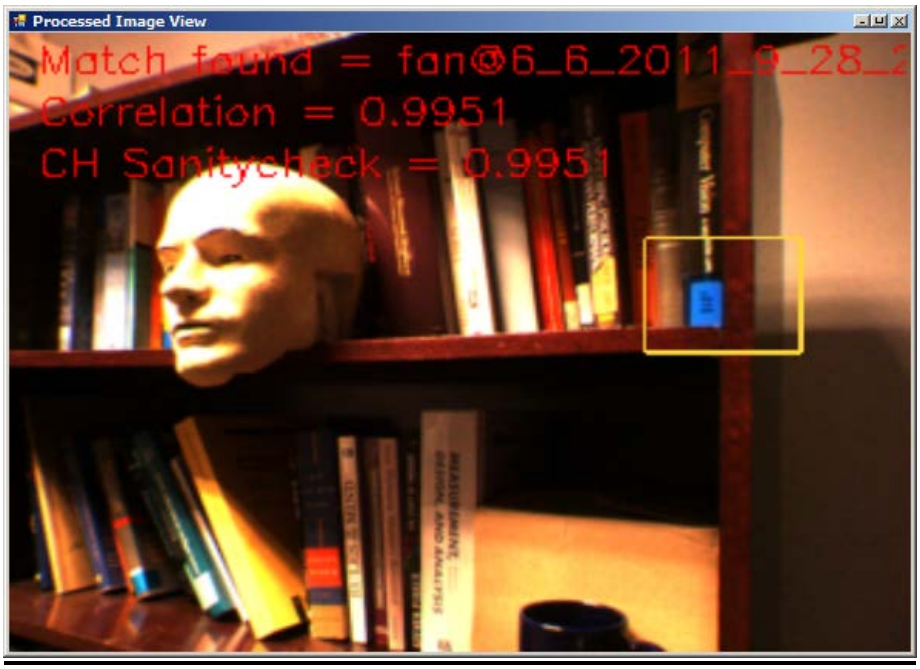


Fig. 73 Color histogram match

Additional Commands

Execs (TemplateMatch quickfindon)
Issuing this command will result in the first match to reach the specified correlation threshold being chosen as a match.

Execs (TemplateMatch quickfindoff)
This command will resume normal searching, and the target with the highest correlation value is chosen. This could mean all targets are examined before a match is made, so the processing could take several seconds.

Execs (TemplateMatch setcorthreshold .99)
This sets the value a correlation must reach before being considered valid. The default value is –1, which will indicate that only the highest correlation will be accepted. Using this in conjunction with the quickfind option sacrifices potentially better candidates but results in a much quicker match. This is sometimes desirable when the number of targets is large.

Execs (TemplateMatch setcolthreshold .99)
This sets the value a color histogram correlation must reach before being considered valid. The default value is –1, which will indicate that only the highest correlation will be accepted. Using this in conjunction with the quickfind option sacrifices potentially better candidates but results in a much quicker match. This is sometimes desirable when the number of targets is large.

Target Facts

Upon subsim initialization, all target subdirectories (targets/, etc.) are scanned, and a fact describing each is created and displayed in the Facts tab of the Goal Edit dialog box, an example of which is shown in Fig. 74.

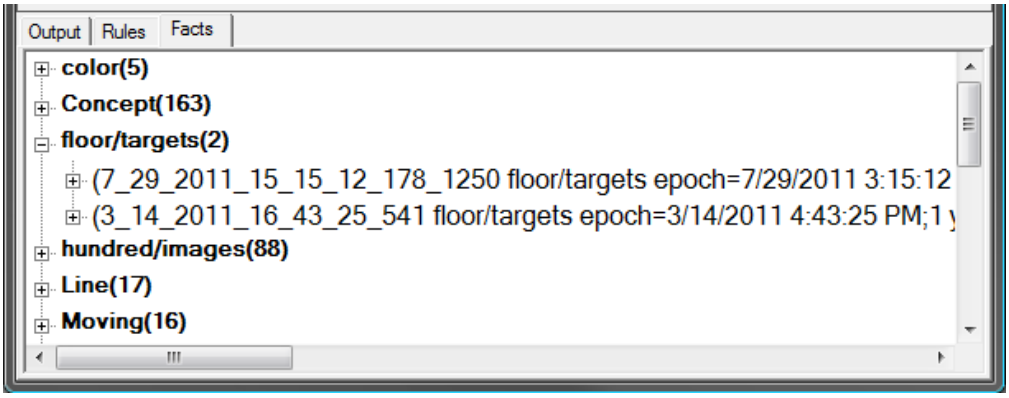


Fig. 74 Target facts in the Facts window

The type of the fact is the name of the subdirectory with the top-level directory appended to the rest of the subdir name (e.g., the subdir name is targets/mugshots/informal so the type is mugshots/informal/targets). The fact itself contains the following fields:

- Name: part of the name of the file of the image; the timestamp portion as described previously.
- Type: as described.
- Year, month, day, hour, minute, second, and millisecond in individual slots (same as the name, only separated).
- Epoch: DateTime object string format of the name (the time that the image was saved).
- Label: first part of the file name (e.g., in file name me@desk-7_8_2011_10_23_0_500_233, the label is me@desk). There is an extra field at the end that represents the distance in millimeters from which the image was captured. If no distance data are available (because the camera was not stereoscopic or it failed to render a depth), that value will be the string NULL.
- Depth: distance from which the image was captured. If no distance data are available (because the camera was not stereoscopic or it failed to render a depth), that value will be the string NULL.

- ImageName: the complete full path name of the file starting with the drive letter.
- ImageMatrix: image data from the file.

7.19 Creating Templates from Other Subsims

As described, subsims are designed to interact with each other, primarily by sending output from one to another. The map-building subsims do this in a daisy chain. Points sends sets of points to the Linesegment subsim, which stitches them together into line segments and then sends them to the Line subsim, which assembles the line segments into lines that are then fed to the Intersect subsim to detect and construct any intersections, and so forth. Other subsims send regions of interest (ROIs)—rectangular squares in the FOV—that are, so far, moments and motion detection. The commands for each are the same as for sending ROIs to the PredatorTLD or Tracking subsims. If you want to send ROIs from these subsims to the TemplateMatch subsim, add their names to the input list in the config file. These facts will be created with a type of <subsimname>/targets (moments/targets, motiondetect/targets, etc.) To differentiate between emitting ROIs to be tracked and ROIs to be captured as targets, each subsim has a CaptureLevel property that can be set to Track, Capture, or TrackAndCapture to notify the recipient subsim of its intention, as shown in the following command:

Execs (<subsimname> setcapturelevel [capture|track|trackandcapture])

Although Template Matching does not generally incur the same memory cost as Image Correlation does over a large number of images (the templates are usually a fraction of the size of a full image), it is conceivable that eventually it can result in an out-of-memory exception due to a large number of templates. If you plan to maintain an extensive cache of targets (several hundred/thousands), you might opt to not preload the images. This will incur some I/O costs, as the images will all have to be read from disk first, but these have been minimal. See Section 7.1 for information on how to specify this and set it as a default if you wish.

7.20 Facial Recognition (Eigenface/Fisherface/LocalBinaryPatternHistograms Methods)

The ability to recognize a face (particularly a friendly one such as the robot's handler) is fundamental to the cognitive abilities of humans and many other mammals, particularly primates. Three such methods provided by the open-source package, openCV, are EigenFaces, FisherFaces, and LocalBinaryPatternHistograms (LBPH), and are available to the robot. The input

into this subsim is currently the FaceRecognition subsim. It detects faces using Haar classifiers and outputs a bounding box to this subsim. In the future, this mechanism could possibly be duplicated in this subsim.

All 3 mechanisms share a common interface and near-identical parameters (LBPH being somewhat different than the first 2). Eigenfaces and Fisherfaces both use a form of component analysis and allow the user to specify the number of elements (faces) to keep. The default is to keep all, and that is the recommendation. LBPH operates quite differently internally and presents the user with the ability to adjust its parameters accordingly. Each allows the user to specify a tolerance/threshold value that will be applied to the prediction of a candidate face. If the distance to the nearest neighbor is greater than the specified threshold, the prediction will result in a –1 (meaning no match was found). For a more complete description of these 3 algorithms and their implementation in openCV, visit

http://docs.opencv.org/trunk/modules/contrib/doc/facerec/index.html

The general flow of operation is as follows:

- Initialize the face recognizer.
- Train the face recognizer either by loading a trained recognizer or dynamically training one through face facts or candidate pictures in a directory specified in a file.
- Optionally, save a recognizer to disk.
- Feed the face recognizer with candidate faces to predict (identify).
- Capture the current result and publish in a fact.

Commands to Initialize a Face Recognizer

Execs (eigenface setmodel Eigen none)
Execs (fisherface setmodel Eigen none)
Execs (lbphface setmodel Eigen none)

These commands will initialize a face recognizer for each of the 3 available models. Only one command is used per subsim instance. To use all 3 at once, specify 3 instances of this subsim in the config file, where they are conveniently named eigenface, fisherface, and lbphface. The names are just those chosen for these examples. You can name the subsim instance anything you want in the config file. Specify “none”, which means each model will be initialized in its default mode, which will use preset default parameters for each.

For EigenFace this means the tolerance is set to MAX_DBL and will keep all of the principal components generated for face prediction.

- NumberOfPCA2Keep: number of principal components (or Eigenfaces) kept for this analysis. Hint: There is no rule as to how many components (read: Eigenfaces) should be kept for good reconstruction capabilities. It is based on your input data, so experiment with the number. Keeping 80 components should almost always be sufficient. (PCA = Principle Components Analysis.)
- EigenFaceConfidence: threshold applied in the prediction.

To specify either or both of these parameters when initializing, use the following commands:

Execs (eigenface setmodel Eigen pcalimits)
This will use the current value of the property NumberOfPCA2Keep and use the default value of EigenFaceConfidence to initialize the model.

Execs (eigenface setmodel Eigen confidence)
This will use the current value of the property EigenFaceConfidence and use the default value of NumberOfPCA2Keep to initialize the model.

Execs (eigenface setmodel Eigen pcalimitsandconfidence)
This will use the current values of the properties NumberOfPCA2Keep and EigenFaceConfidence to initialize the model.

Execs (eigenface seteigenfacepcalimits 10)
This will set the value of the property NumberOfPCA2Keep to 10.

Execs (eigenface seteigenfaceconfidence 16000)
This will set the value of the property EigenFaceConfidence to 16,000.

For FisherFace this means the tolerance is set to MAX_DBL, and the defaults will keep all of the linear discriminant components generated for face prediction.

- NumberOfFF2Keep: The number of linear discriminat components (or FisherFaces) kept for this analysis with the FisherFaces criterion. It is useful to keep all components, which means the number of your classes (c); that is, subjects and persons you want to recognize. If you leave this at the default (0) or set it to a value ≤ 0 or $\geq$ (c-1), it will be set to the correct number (c-1) automatically.
- FisherFaceConfidence: Threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns –1.

To specify either or both of these parameters when initializing, use the following commands:

Execs (fisherface setmodel Fisher fflimits)
This will use the current value of the property NumberOfFF2Keep and use the default value of FisherFaceConfidence to initialize the model.

Execs (fisherface setmodel Fisher confidence)
This will use the current value of the property FisherFaceConfidence and use the default value of NumberOfFF2Keep to initialize the model.

Execs (fisherface setmodel Fisher pcalimitsandconfidence)
This will use the current values of the properties NumberOfFF2Keep and FisherFaceConfidence to initialize the model.

Execs (fisherface setfisherfacefflimits 10)
This will set the value of the property NumberOfFF2Keep to 10.

Execs (fisherface setfischerfaceconfidence 16000)
This will set the value of the property FisherFaceConfidence to 16,000.

For LBPH this means the radius will be set to 1, the neighbors value set to 8, the gridx value set to 8; and the gridy value set to 8.

- LBPHRadius: radius used for building the Circular Local Binary Pattern. This is the distance from the center of a grid to its neighbors in number of grid cells.
- LBPHNeighbors: number of sample points to build a Circular Local Binary Pattern from. An appropriate value is to use 8 sample points. The more sample points you include, the higher the computational cost.
- LBPHGrid_x: number of cells in the horizontal direction. A common value used in publications is 8. The more cells, the finer the grid and the higher the dimensionality of the resulting feature vector.
- LBPHGrid_y: number of cells in the vertical direction. A common value used in publications is 8. The more cells, the finer the grid and the higher the dimensionality of the resulting feature vector.
- LBPHThreshold: threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns –1.

To specify any of these parameters when initializing, use the following commands:

Execs (lbphface setmodel LocalBinary radius)

This will use the current value of the property LBPHRadius and use the default values of the rest of the parameters.

Execs (lbphface setmodel LocalBinary neighbors)
This will use the current values of the properties LBPHRadius and LBPHNeighbors and use the default values of the rest of the parameters.

Execs (lbphface setmodel LocalBinary gridx)
This will use the current values of the properties LBPHRadius, LBPHNeighbors, and LBPHGrid_x and use the default values of the rest of the parameters.

Execs (lbphface setmodel LocalBinary gridy)
This will use the current values of the properties LBPHRadius, LBPHNeighbors, LBPHGrid_x, and LBPHGrid_y and use the default values of the rest of the parameters.

Execs (lbphface setmodel LocalBinary threshold)
This will use the default values of the properties LBPHRadius, LBPHNeighbors, LBPHGrid_x, and LBPHGrid_y and use the current value of the rest property LBPHThreshold.

Execs (lbphface setlbphradius 10)
This will set the value of the property LBPHRadius to 10.

Execs (lbphface setlbphneighbors 16)
This will set the value of the property LBPHRadius to 16.

Execs (lbphface setlbphgridx 16)
This will set the value of the property LBPHGrid_x to 16.

Execs (lbphface setlbphgridy 16)
This will set the value of the property LBPHGrid_y to 16.

Execs (lbphface setlbphthreshold 255)
This will set the value of the property LBPHThreshold to 255.

Commands to Train a Face Recognizer

Execs (eigenface trainmodel)
Execs (fisherface trainmodel)
Execs (lbphface trainmodel)

These commands will train a face recognizer using a couple of different methods. First, photos in a directory that are tabulated in an input file, here named train.dat, will be used. The file contains lines of text that give the name and location of the image (a jpg file), the numeric label of each image (each image of the same face

has the same label because the algorithm requires an integer as a label), and the corresponding actual name of the candidate. In the following case, the location is a subdirectory of the working directory called facecandidates, which contains subdirectories named 0001 to 0051.

Use the following commands in the following sequence:

Execs (lbphface settrainingfilename train.dat)
Execs (lbphface trainmodel)

Train.dat contains the following, showing the last 2 face candidates of 10 pictures each from a pool of 51 faces:

```
facecandidates/0050/00501001.JPG;50;Face-50
facecandidates/0050/00501010.JPG;50;Face-50
facecandidates/0050/00501020.JPG;50;Face-50
facecandidates/0050/00501029.JPG;50;Face-50
facecandidates/0050/00502011.JPG;50;Face-50
facecandidates/0050/00502020.JPG;50;Face-50
facecandidates/0050/00503010.JPG;50;Face-50
facecandidates/0050/00504011.JPG;50;Face-50
facecandidates/0050/00504020.JPG;50;Face-50
facecandidates/0050/00505010.JPG;50;Face-50
facecandidates/0051/Image1.jpg;51;OPERATOR
facecandidates/0051/Image10.jpg;51;OPERATOR
facecandidates/0051/Image2.jpg;51;OPERATOR
facecandidates/0051/Image3.jpg;51;OPERATOR
facecandidates/0051/Image4.jpg;51;OPERATOR
facecandidates/0051/Image5.jpg;51;OPERATOR
facecandidates/0051/Image6.jpg;51;OPERATOR
facecandidates/0051/Image7.jpg;51;OPERATOR
facecandidates/0051/Image8.jpg;51;OPERATOR
facecandidates/0051/Image9.jpg;51;OPERATOR
```

Alternately, face data can be loaded into facts for training using the following command:

Execs (lbphface loadfacesintofacts facecandidates)

This will result in each face loaded into a fact containing the following information:

- Name of the jpg picture without the extension .jpg
- Type: facecandidate
- Slot OpName: name of the subdir (under facecandidates) where the jpg is located. In the described cases, they would be 0050 or 0052. To make it more useful and readable, the subdirs should be named according to the subject contained therein, such as Troy or Sean.

- Slot image: contains the image data wrapped in a class instance that holds an IplImage of the jpg.
- Slot label: contains the integer label described previously. It is the same for each jpg in a subdir and is incremented upon moving to the next subdir.

With an uninitialized model, executing the following goal will result in the model being trained with these facts:

```
goal LBPHFactTrain
{
	rule LBPHFactTrain ()
	{
		Execs (lbphface setmodel LocalBinary none)
		Execs (lbphface trainfromfactson)
		Execs (lbphface trainmodel)
		Execs (lbphface trainfromfactsoff)
		Quit (true)
	}
}
```

Facts can also be loaded by using the camera to take snapshots of a candidate in one of 2 ways. First, the subsim can automatically take pictures of a candidate once his/her face is aligned with a rectangle displayed in the center of the image.

For example, if the following goal is executed,

```
goal CaptureCandidateSEAN
{
	rule CaptureCandidateSEAN()
	{
		Execs(lbphface settrainingcandidate SEAN 1 10)
		Execs (lbphface capturemodeon)
		Execs (lbphface begincapture)
		Execs(lbphface capturemodeoff)
		Quit (true)
	}
}
```

the command

Execs (lbphface settrainingcandidate SEAN 1 10)

sets the candidate name to SEAN, the candidate label (internal to the model) to 1 and the number of images to capture to 10.

Execs (lbphface capturemodeon)
This will place the subsim in capture mode.

The command

Execs (lbphface begincapture)

will start capturing images as defined by the red square overlaying the face. The subject would ideally turn their head slowly from side to side and up and down to get a variety of aspects, and the subsim will pause 2 s between each frame and stop collecting when it reaches the limit, set to 10 in the first command.

Figure 75 shows what the image should look like during this process.

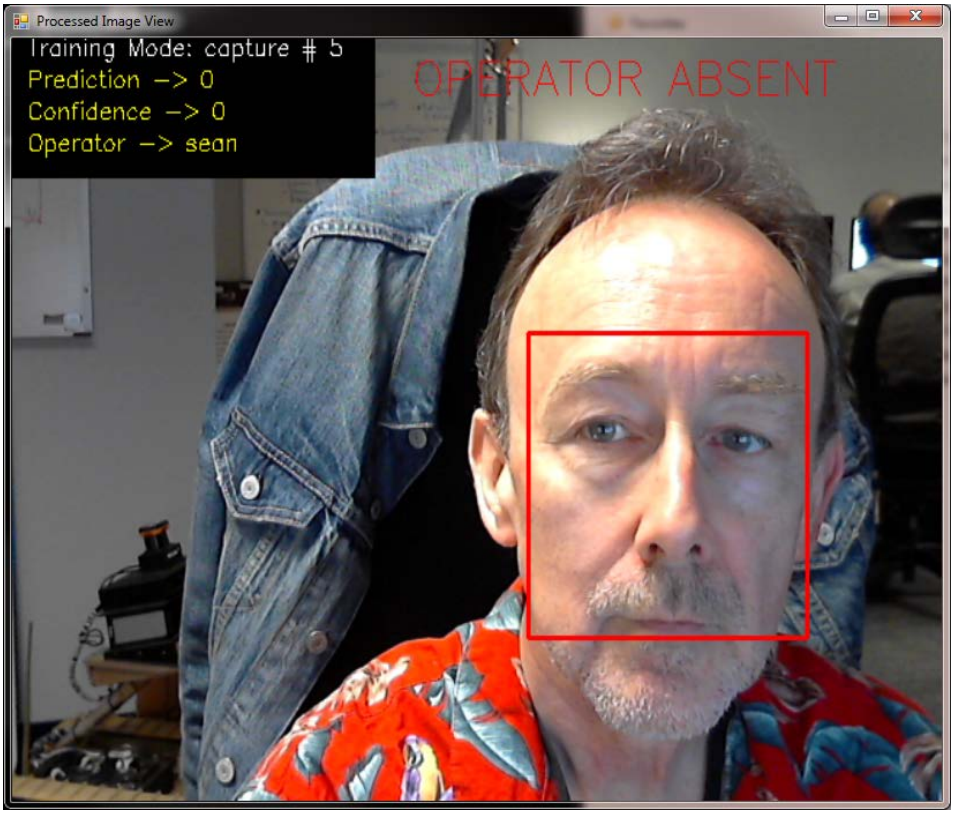


Fig. 75 Capturing a subject for training

To manually snap pictures, do not issue the command to begin capture. Rather, when the rectangle turns green, issue the following command:

Execs (lbphface snap)

This will cause the subsim to take a snapshot and allow the user to control the angle of the face when the picture is taken. The automatic mode will take several shots in rapid succession, which will not allow for much variance in posing. It is advantageous to deliberately vary the pose for greater accuracy in prediction after the model is trained with these pictures.

A model that has been initialized and trained can be retrained by repeating these steps to do the initial training. This will result in completely retraining the model. In the case of the LBPH model, you can use the following command:

Execs (lbphface updatemodel)

This will enhance the model with the additional data.

Commands to Load and Save a Face Recognizer

Optionally saving and loading a model that has been trained can save a great deal of time and provide a good deal of flexibility in using this subsim. To save a trained model, issue the following command:

Execs (lbphface savemodel [filename])

This will save the model using the name specified by filename with the extension .yaml. If no name is specified, the current name contained in the property ModelSaveFileName will be used. If this property is set to None (default value), the save will not occur. Upon saving the model, a parallel file will be saved containing the readable names of each face trained. It will be the same name as the model's save file with the additioinal extension of .names.

A saved model can thus be reloaded using the following command:

Execs (lbphface loadmodel [filename])

This will load the model saved in the file specified by filename. If no filename is specified, the name contained in the property ModelLoadFileName will be used unless it is set to None, which is the default value. The corresponding file with the names will also be loaded and used to display the model's predictions upon being given faces to identify.

Recognizing (Predicting) a Face Using a Trained Face Recognizer

Once the model is trained, to get it to attempt to identify a face, it needs to have a face candidate sent to it by the Face subsim (described in the following). This accomplished by setting up the config file as

```
     <add         name="faces"         type="Brain.CSubSimProcessorFaces"
assembly="Brain" run="true">

        <properties>
          <add property="UseLocalVFWDevice" value="True" />
          <add property="TrainingInterval" value="3" />
          <add property="TrackType"
value="haarcascade_frontalface_alt_tree.xml" />
          <add property="TrackFace" value="True" />
          <add property="SanityChecking" value="True" />
          <add property="Operator" value="SEAN" />
          <add property="OperatorAbsence" value="0" />
          <add property="FauxFaceThreshold" value="0" />
          <add property="DebugLevel" value="Verbose" />
          <add property="ClassifyVigilance" value="0.8" />
          <add property="TrainingVigilance" value="0.7" />
          <add property="ResultFactType" value="arg" />
          <add property="CompareToMean" value="True" />
          <add property="EigenFeatures" value="13" />
          <add property="CaptureLevel" value="None" />
          <add property="InscribeROI" value="True" />
        </properties>
      </add>

      <add name="eigenface"
type="Brain.CSubSimProcessorFacialRecognition" assembly="Brain"
run="true" input="faces"/>
      <add name="fisherface"
type="Brain.CSubSimProcessorFacialRecognition" assembly="Brain"
run="true" input="faces"/>
      <add name="lbphface" type="Brain.CSubSimProcessorFacialRecognition"
assembly="Brain" run="true" input="faces"/>
```

Figure 76 shows what the screen should look like if the operator (here specified as Sean) is present.

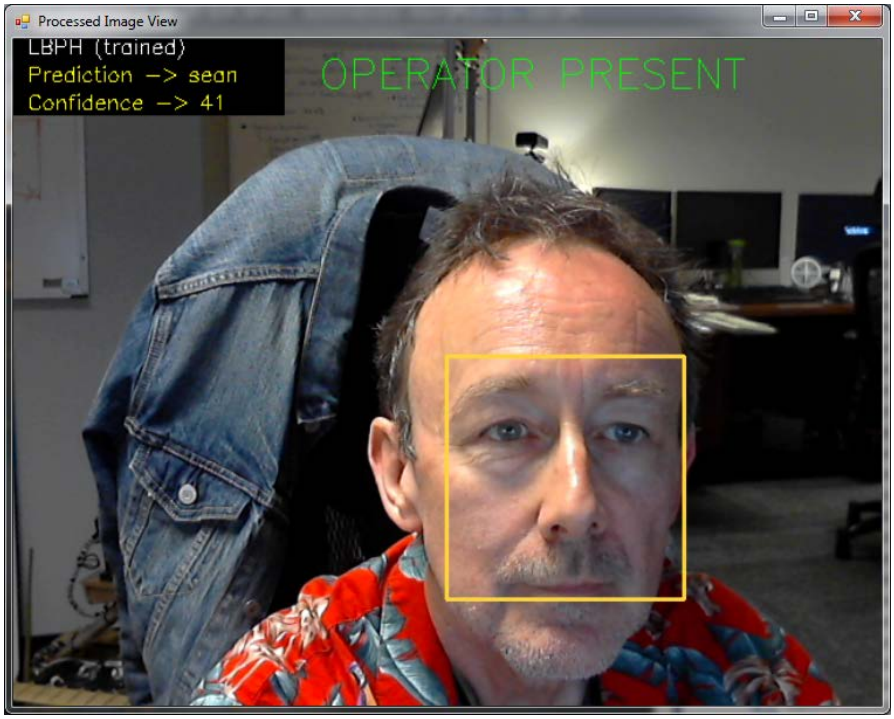


Fig. 76 Recognizing subject as the operator of the robot

Figure 77 shows what the image should look like if the operator is absent.

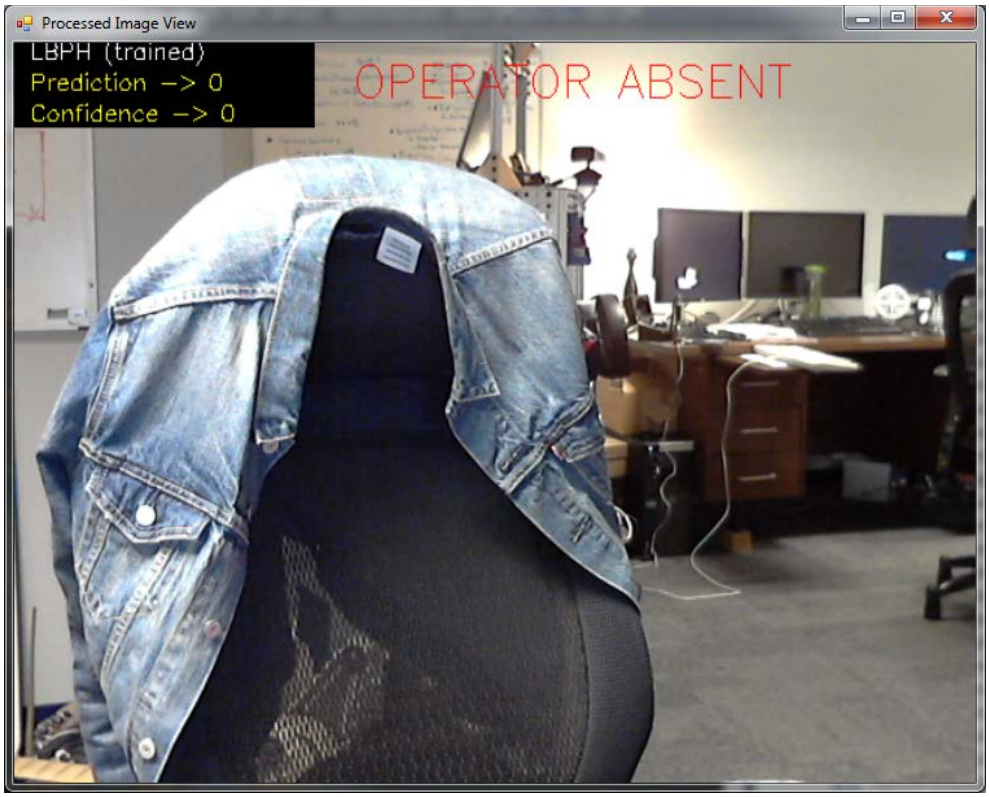


Fig. 77 Reporting operator is absent

Publising the Result in a Fact

Each time (frame by frame) a prediction is made as to who the face is, the result is placed in a State fact that has been added to the mind upon subsim startup (Fig. 78).

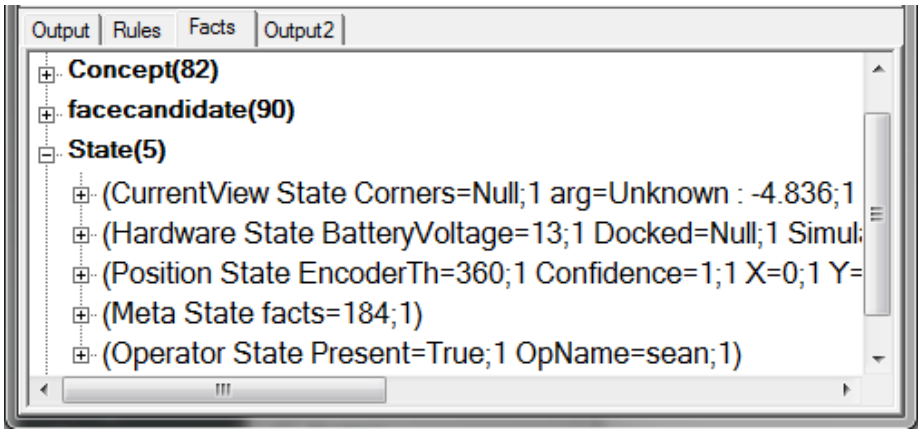


Fig. 78 Operator State fact showing operator has been identified

Collecting Statistics

To verify that the subsim has identified the candidate face is who it says it is, a prescribed percentage of the time (how many frames correct given a prescribed interval of frames) statistics can be gathered. The following goal can be run to accomplish this:

```
goal CollectSeanStatsLBPH

{
        rule CollectSeanStatsLBPH()
        {
                Execs(lbphface setstatisticsfilename Sean)
                Execs(lbphface setstatisticsgroundtruthname Sean)
                Execs(lbphface setstatisticsframelimit 100)
                Execs(lbphface startgatheringstatistics)
                Quit(true)
        }
}
```

The filename is set using the command setstatisticsfilename, which will take Sean and append a timestamp to it plus the suffix .csv, which can be opened by Excel. The contents will be a sequence of readings, one per frame, which will be ground truth, predicted face, label number, and confidence level. While the stats are being collected, the screen will look something like the image shown in Fig. 79.

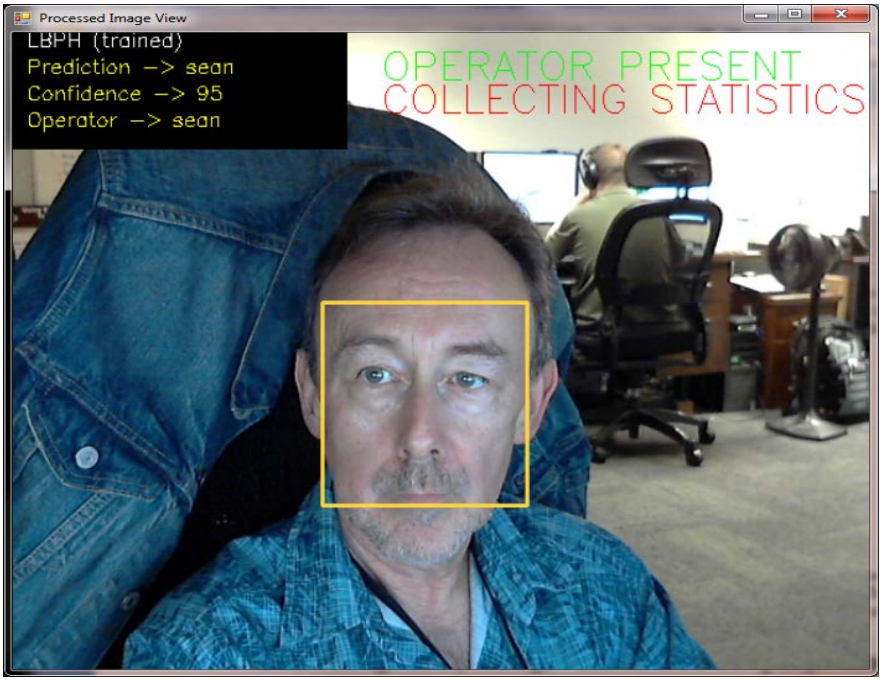


Fig. 79 Collecting Statistics screen

As soon as the collection period is finished, Collecting Statistics will disappear.

7.21 Facial Recognition (Average Face Method Using EigenFace)

Using a 3-pronged approach, this is another way the robot has of identifying, learning, and remembering faces. These 3 elements are the following:

- Haar Cascade feature detectors trained to identify facial elements such as eyes, noses, lips, ears, and the like. This step identifies candidate faces. (Google these or read about them in "OpenCV – Computer Vision" with the OpenCV library.) If the Haar classifier file being used is not present, the error box shown in Fig. 80 will be displayed.

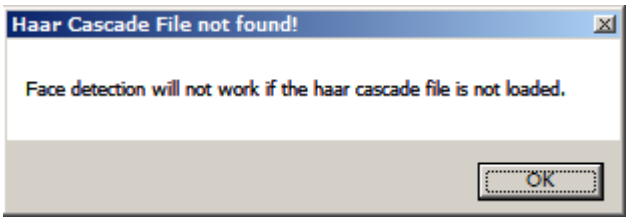


Fig. 80 Missing Haar file message box

- EigenFace facial discrimination code extracts salient features from hundreds of random sample faces into an average face for comparison with the features extracted from an unknown face identified by the Haar classifiers. This step provides a basis for comparisons.
- ARTMap neural network takes the data from the EigenFace recognition process and uses them to "learn" individual facial signatures. This step allows for persisting recognized faces.

The general algorithm is as follows:

- Train the EigenFace facility with a reasonable number (about 300) of sample faces. This will create a "space" for comparing candidate faces as they are detected by the Haar classifiers. A file called train.txt must be present in the executable directory with contents similar to the following very sample number:

```
1 C:/COLORFETADJUSTED3_noglasses/images/image_1.ppm
2 C:/COLORFETADJUSTED3_noglasses/images/image_1000.ppm
3 C:/COLORFETADJUSTED3_noglasses/images/image_1002.ppm
4 C:/COLORFETADJUSTED3_noglasses/images/image_1003.ppm
5 C:/COLORFETADJUSTED3_noglasses/images/image_1004.ppm
6 C:/COLORFETADJUSTED3_noglasses/images/image_1005.ppm
7 C:/COLORFETADJUSTED3_noglasses/images/image_1006.ppm
8 C:/COLORFETADJUSTED3_noglasses/images/image_1007.ppm
9 C:/COLORFETADJUSTED3_noglasses/images/image_1008.ppm
10 C:/COLORFETADJUSTED3_noglasses/images/image_1009.ppm
11 C:/COLORFETADJUSTED3_noglasses/images/image_101.ppm
12 C:/COLORFETADJUSTED3_noglasses/images/image_1010.ppm
13 C:/COLORFETADJUSTED3_noglasses/images/image_1011.ppm
```

```
14 C:/COLORFETADJUSTED3_noglasses/images/image_1012.ppm
15 C:/COLORFETADJUSTED3_noglasses/images/image_1013.ppm
16 C:/COLORFETADJUSTED3_noglasses/images/image_1014.ppm
17 C:/COLORFETADJUSTED3_noglasses/images/image_1015.ppm
18 C:/COLORFETADJUSTED3_noglasses/images/image_1017.ppm
19 C:/COLORFETADJUSTED3_noglasses/images/image_1018.ppm
20 C:/COLORFETADJUSTED3_noglasses/images/image_1019.ppm
21 C:/COLORFETADJUSTED3_noglasses/images/image_102.ppm
22 C:/COLORFETADJUSTED3_noglasses/images/image_1020.ppm
23 C:/COLORFETADJUSTED3_noglasses/images/image_1022.ppm
24 C:/COLORFETADJUSTED3_noglasses/images/image_1023.ppm
25 C:/COLORFETADJUSTED3_noglasses/images/image_1025.ppm
```

Naturally, the files listed in this file need to exist in the locations specified. Each of the training files must be of the same height and width (e.g., 120 ×120 pixels).

Once this is done, an XML file called facedata.xml will have been created and can be used in future runs without retraining. It is placed in the executable's folder and until removed will serve that purpose of training the EigenFace. This process will also create an image file called avim.bmp, which contains a face representative of an average of all the features contained in the training files. It will be used by the face detector (Haar) to weed out any false positives that are detected. Face detection often results in candidates found where there are none. All of the candidate faces will be compared with this file using Pearson's correlation algorithm mentioned earlier in this manual. A high enough score means that the area in question is enough like a face to safely assume it is one. This candidate will be passed on to the classification function. Bear in mind that this process is not foolproof either; a few nonfaces will get through from time to time. This is primarily a tool to limit wasted processing time. Note that the face detector will work regardless of whether or not the EigenFace is trained.

The following goal will train the EigenFace and turn on the classification code:

Execs (faces artfaceson)

For those who do not have access to such a face database, the robot code comes bundled with the facedata.xml file and will automatically detect its presence.

- The next phase is classification and recognition. Initially, the EigenFace protocol calls for the classification to be based on a comparison of the Eigenvalues extracted from a candidate face and those imprinted in the space created from initial training. This is useful for candidate faces that were included in the training set but generally useless for those not included. After several alternative measurements were considered, one was determined to yield the best results. The mean of all the faces represented in the space is computed, and the Eigenvalues gleaned from the candidate face are compared with that. The difference between them is calculated and

fed to an ARTMap neural net, which will return an initial classification of Unknown since it has not been trained yet. Figure 81 is an example of a screen full of untrained faces. (For convenience, we use pictures of several people rather than streaming them live. Live faces seem to train a bit better but many things affect the accuracy, primarily scale and lighting.)

Fig. 81 Unknown face candidates

To teach the ARTMap which face belongs to whom, a rectangle is first drawn around it (Fig. 82).

Fig. 82 Face selected for training

Then the following goal specifying the name of the person is executed:

Execs (faces setsubject Eric)

Figure 83 shows what the screen then looks like.

Fig. 83 Selected face after training, Eric

As you can see, the face now has a name, Eric_0, and a confidence level returned by the ARTMap. The "0" appended to the name represents the first such face trained. As other faces in the scene are trained, the number increments, making it possible to train the ARTMap with the same name for other individuals or a different view of the same individual. The confidence level (in Eric's case, 0.9989) indicates the ARTMap's recognition score. Note that the others went from 0 to various scores: 0.7344, 0.7112, 0.7827, and so on. This shows that the ARTMap, having been trained on a face, now attempts to categorize the faces and renders a score on each face it is presented with. The face of the actual Eric gets the highest score as is desirable, and the rest get lower scores. Since they fall below the vigilance threshold for classification (seen at the top left of the screen as 0.84), they are still classified as Unknown. Figure 84 shows what the screen looks like, fully trained.

Fig. 84 All faces after training

While the differences in lighting changes and scale (how large or small the face is in the image) may seem small to the human observer, they are enormous to the software that detects a candidate face and classifies them after training. Therefore, a lot of misclassification can occur. One way to mitigate this is to train the ARTMap on the same face in different lighting, scale, and position in the image. If, for instance, the Eric face were to be trained this way, it could result in faces named Eric_0, Eric_1, Eric_2, and so on. Each name is unique to the ARTMap, but the observer would see that the name was the same regardless of the suffix. Another method (under development) is to have the face recognition engine self-train by automatically assigning an a priori name to each face such as Face#0, Face#1, and so on, creating a fact for each of them carrying the name and the location of the face. The idea is to mitigate the possibility of misclassification by reconciling the face and locations with the name chosen for it. When this feature is fully operational, it will be added to this manual.

The values appearing in the upper left corner of the images are the following:

T-Vigilance: This number controls how rigid the ARTMap is in creating categories. A lower number (0, in this case) results best in future classification.

C-Vigilance: This number controls how rigid the ARTMap needs to be to correctly identify a trained face. This higher the number, the higher the score needed to accurately classify a face. As we have seen, once a face is trained, it should get the highest score, but the ARTMap will attempt to rate all the faces it is presented with. The lower the score on incorrect candidates, the better.

F-Threshold: This number controls the filtering of false-positive facial candidates. The Haar classifiers are very good but can be fooled, as they do not find actual facial features. Instead, they identify features they have been trained to recognize as statistically viable members of a human face. Statistical models tend to have margins of error, and this one is no exception. Thus, an image might contain many false positives as Fig. 85 shows.

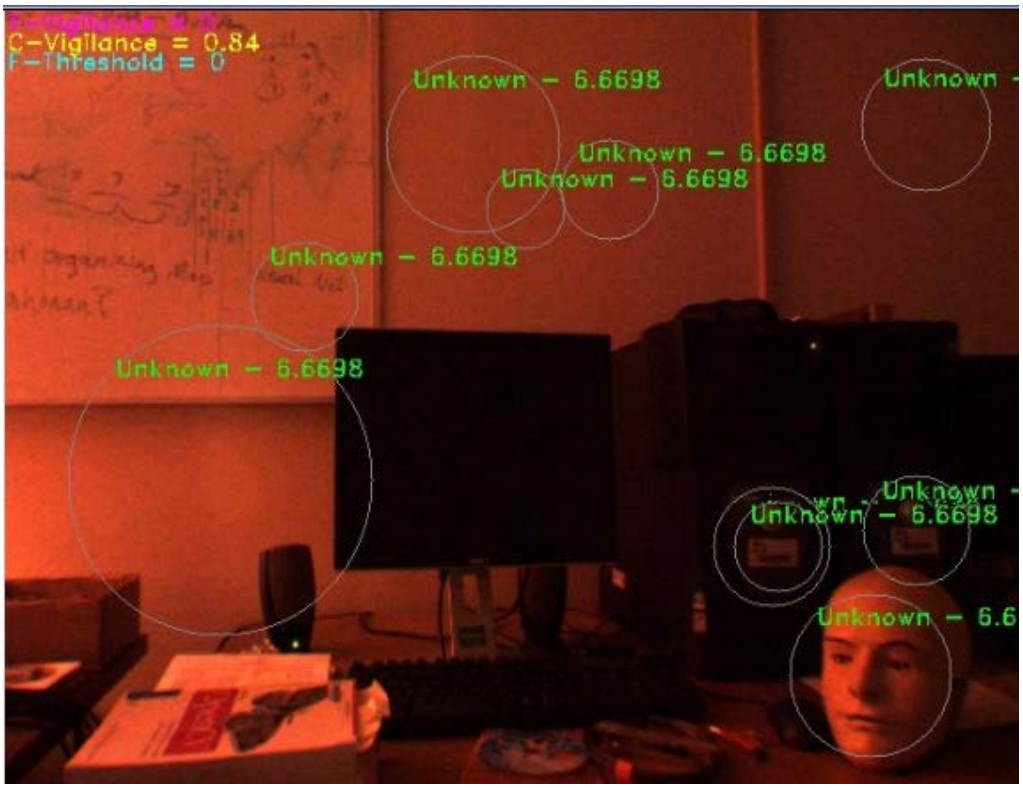


Fig. 85 False-positive faces

In this case, the parameter F-Threshold is set to 0. The parameter can be tweaked until most or all of the false faces are gone and only a legitimate face is presented, as shown in Fig 86.

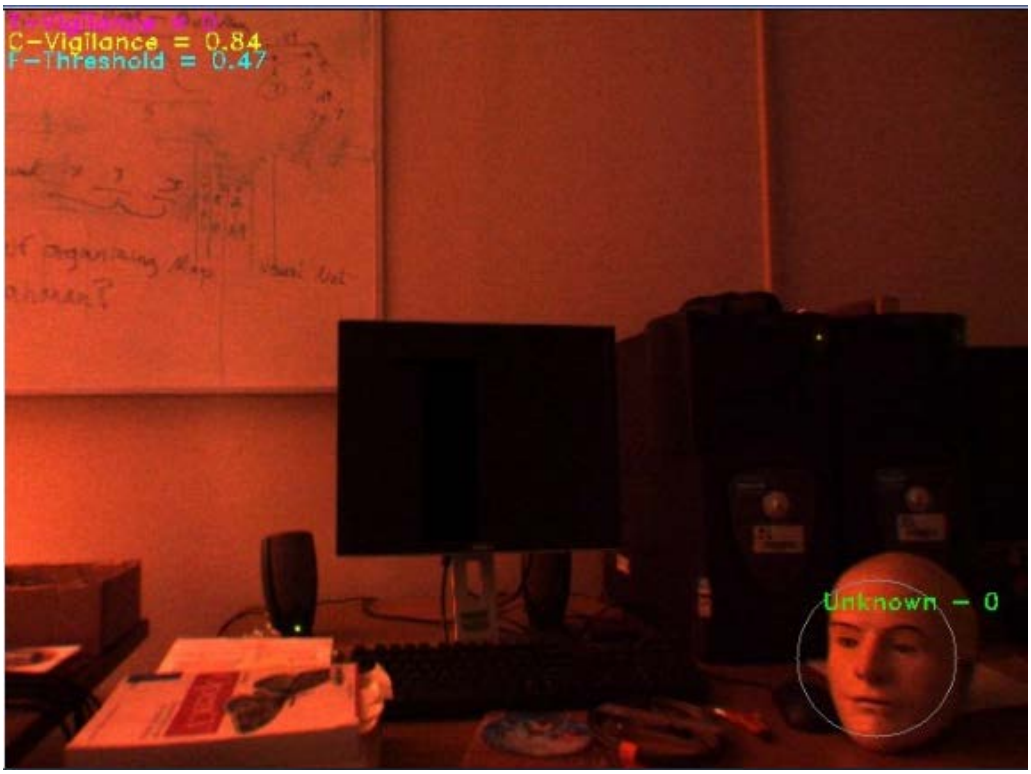


Fig. 86 False positives almost eliminated

In this scene we see that the F-Threshold is set to 0.47 and the faux faces have been vanquished. This setting represents a correlation value threshold that is compared with the score that an average face image (generated by the EigenFace training process) and a candidate face must exceed when the 2 are correlated using Pearson's correlation algorithm. The resulting face is shown in Fig. 87.

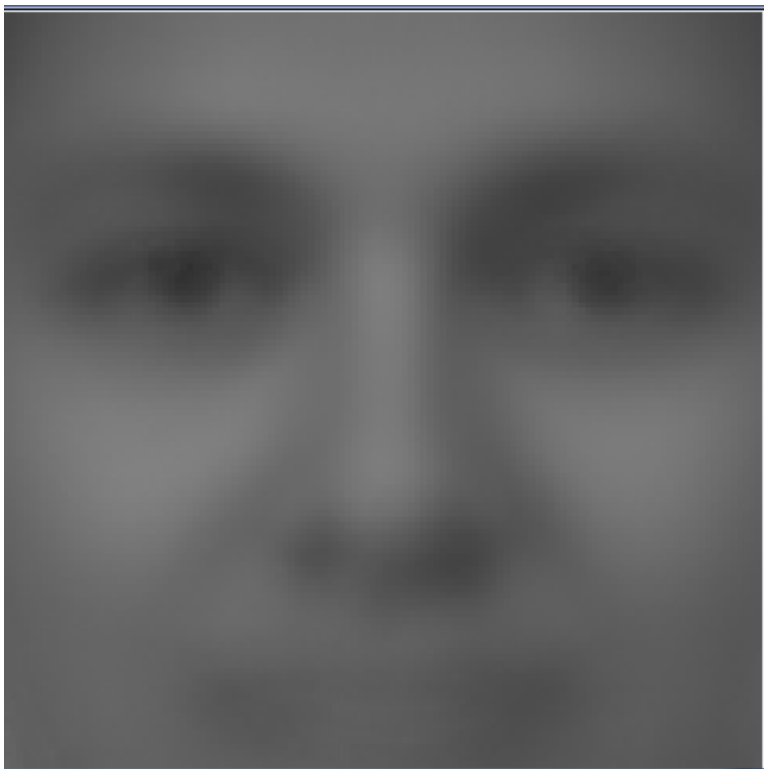


Fig. 87 Average face generated from EigenFace training

Since the average face is also a variable in the mix, it too can change. Currently, the name of the file is avimg.bmp. To change the file, simply put another in its place under the same name.

To be fair, these images were snapped in poor lighting and inadequate white balance. This contributed greatly to the emergence of nonfaces as candidates. Best results can be obtained using adequate lighting and prewhite balancing the camera. Figure 88 shows the same face (Mr Head) in a different setting with good lighting and balance.

Fig. 88 False positives in good lighting

No attempt was made to remove false positives.

Additional Commands

The ARTMap that categorizes the faces it learns with the EigenFace component can be saved, loaded, and initialized. The following are the commands you would specify in the consequent of a goal for these faces.

- Saving: An ARTMap can be saved using a command specified as *Execs (faces saveartfaces filename)*, where the filename is the name given to the serialized portion of the ARTMap. The name of this file, the number of features, the vigilance value, and the individual categories and their supervisor numbers will all be stored in a file called ARTMapTopologyFaces.xml.

  Loading: An ARTMap (previously trained) can be loaded using a command specified as *Execs (faces loadartfaces)*, where the serialized portion of the ARTMap, the number of features, the vigilance value, and the individual categories and their supervisor numbers will all be loaded from the file ARTMapTopologyFaces.xml. It is important that the file contained categories from previous training or an exception will be thrown and the program will need to be restarted.

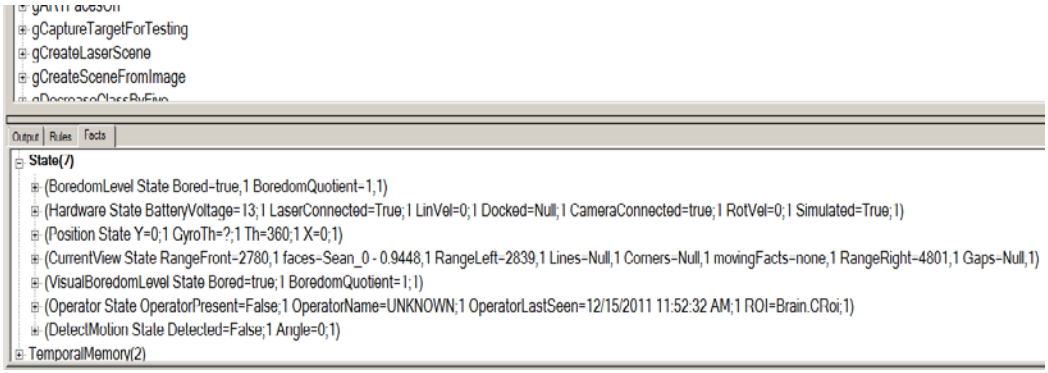
- Initializing: To create a new ARTMap for face categories, use the command *Execs (faces initartfaces features)*. The features parameter is an integer greater than 1, which denotes the number of features the ARTMap will expect from the input. Currently, this is set to 10 in the code.

Faces can also be handed off to the PredatorTLD subsim using the following command:

Execs(faces trackfaces)

This command sends a single face candidate (a bounding box inscribed in the circle denoting a detected face) to the PredatorTLD subsim, which begins tracking it.

Additionally, when faces are detected, they are reported in the CurretView/State fact with the slot faces. This list can be referenced in a goal for productions that make use of recognized faces, as in Fig. 89.



Note: Do not save or load an artmap that has not been trained. This can cause the program to throw an exception, which will require a restart.

Fig. 89 Face list in CurrentView/State Fact

7.22 Moments Detection

Moments, in this context, refer to defining attributes of contours detected by openCV functions (Fig. 90). These contours are essentially outlines of areas in an image that are more or less homogeneous and/or consistent. A baseball sitting on a table would, for instance, generate a contour that is more or less round, thus outlining it and tagging it as an object that can be segmented (isolated) in the image and perhaps recognizable and learnable to the robot and its neural networks. Similarly, a doorway might generate a rectangular set of contours, thus defining where the door is in the image and its dimensions. These values (7 are generated for each contour) can be considered reasonably unique (a set of moments from a circular object being distinct from those of a rectangular object) and thus suitable for training a neural network such as the ARTMap. While not as precise (compared with more-instance-based data from the laser or template matching), it is a valuable

part of the combined effort of the many feature-detection facilities available in the robot.

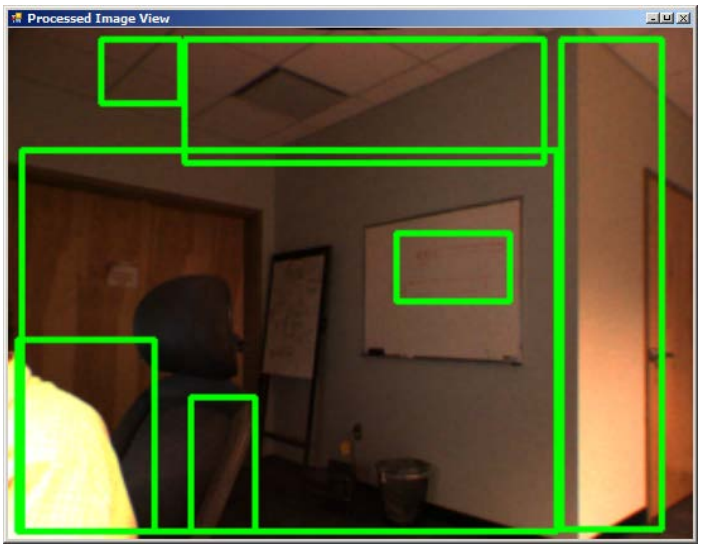


Fig. 90 Example of Moments Detection view

Commands

Execs (moments createscene office building)
Issuing this command will result in the current image being saved to a jpg file possibly named office-6_12_2012_10_32_30_588.jpg (the timestamp portion of the name will vary) in a subdirectory of the Moments directory called Building. The path to the file would be the following:

moments/building/office-6_12_2012_10_32_30_588.jpg

Execs (moments sortmoments)
Issuing this command will result in the currently detected moments being sorted in ascending order by area (width*height in pixels) before being sent to the Tracking subsim.

Execs (moments trainartmapfromimage corner office-6_12_2012_10_32_30_588.jpg)
This command results in the file images/building/office-6_12_2012_10_32_30_588.jpg being loaded, its moment(s) value(s) extracted and fed to an ARTMap trained to recognize said moments as the classification corner.

Execs (moments trackmomentson)
This command sends the regions detected to the Tracking subsim for tracking. The Tracking subsim will accept as many regions as it is set to accommodate. Until the command to stop doing this is issued, the subsim will continue to send regions to

the Tracking subsim. Detected regions are reported as facts of type Arg and name glob_xxx, where xxx is the number of the detection (starting at 0).

Execs (moments trackmomentsoff)
This command stops sending regions to the Tracking subsim. Regions being tracked by the Tracking subsim will continue to be tracked until terminated in that subsim.

7.23 Color Detection/Tracking

The ability to find regions of more or less the same color is available in this subsim. Regions of (more or less) homogenous hue that match are detected, delineated, sorted according to area (if desired), and sent to the Tracking subsim (if desired). The number of hits can be adjusted by setting area and perimeter limits as well as hue, saturation, and brightness biases. All single-color detections are achieved by converting red–green–blue (RGB) to hue, saturation, and value (HSV) format values.

The subsim can currently detect color "blobs" using 2 different methods: pixel-by-pixel comparison of the target hue and histogram detection. Target-hue detection allows for the detection of a single solid color. Histogram detection allows for detecting a collection of colors such as skin tones, heterogeneous patterns, or a range of colors covering a hue spectrum like orange to red or blue to green using histograms. The results are reported on the screen by bounding boxes for both single color and histogram detections and with perimeters drawn around the regions if using histogram detection. They are also reported as facts much the same as motion detections and moments are and thus can be made use of in the production system.

7.32.1 Single-Color Detection

The system comes with an assortment of predefined colors with interesting names like Alice Blue, Chartreuse, Cornsilk, and Fuchsia as well as the standard red, green, blue, and the like. These colors are predefined, and a file containing their definitions is created and placed in the colors folder in the working directory of the executable. This file is called colorFile.txt and is generated and saved automatically if it does not exist upon startup of the subsim. This format is an ASCII-text-based file with the name of the color, RGB values, HSV equivalent values, and HSV bias values, all separated by spaces. HSV bias values are tolerances for these components of the color. Detections occur when the value of each pixel falls within these tolerances when they are applied to the hue, saturation, and brightness components. Single colors to detect can be specified in the following ways.

Eyedropper

The canonical eyedropper from virtually all image-processing suites (Photoshop, PaintShop Pro, etc.) can be used to select a color in the current image by first selecting the Color subsim from the right-click menu on the image. At this point, left-clicking on the image will select the color in the pixel that the mouse was pointing to and begin detecting it and showing such detections on the screen (Fig. 91). The status bar at the top of the screen shows (left to right) a small square showing a patch of the current color, the RGB values of the color, the HSV values of the color, and the bias values for determining the validity of a given pixel in the color search. The word “Eyedropper” appears as the color name.

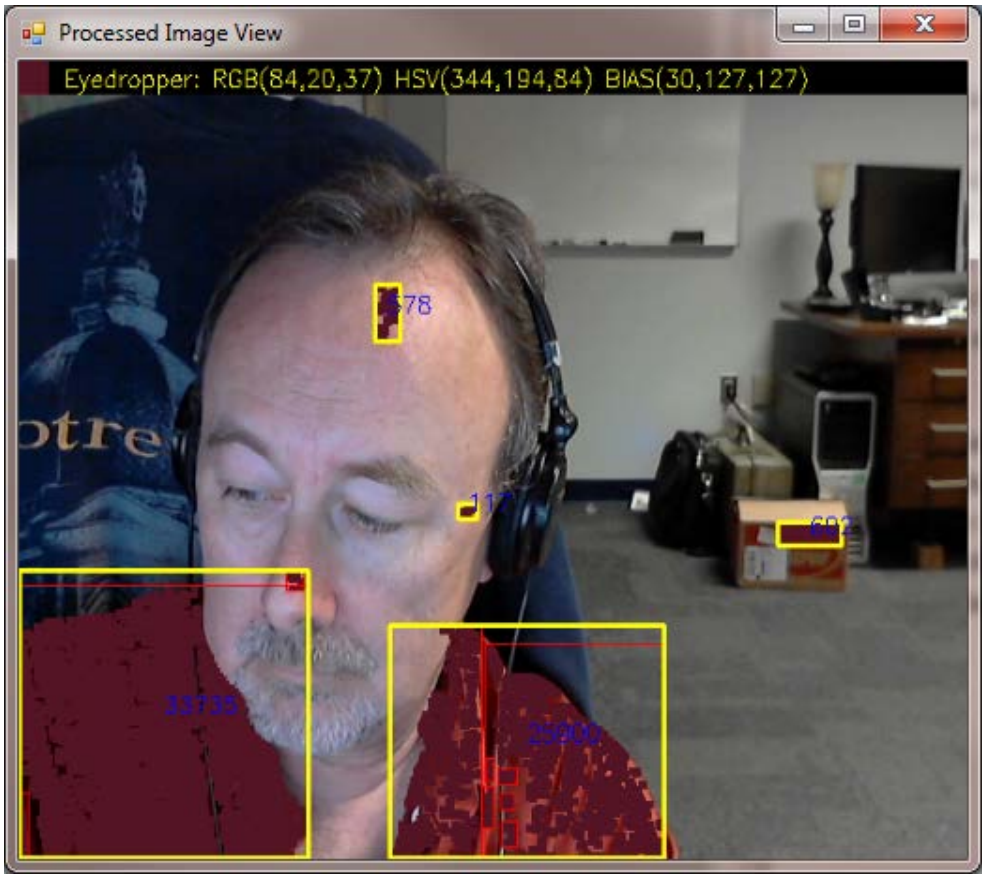


Fig. 91 Result of left-click on subject’s red shirt

Color Picker

The color picker can be invoked by right clicking on the processed image panel and selecting Set/Adjust Colors. Using the mouse to manipulate the square pointer in the color wheel or the trackbars to select the color, the results show up instantly on the processed image screen. Doing this will result in colorPicker being displayed as the current color. The color picker also presents a drop-down selection box

populated by all the colors currently known to the robot. When a color is selected from the drop-down listbox, it becomes the current color and its name is displayed as the current color. You can modify the chosen predefined color (dark red in Figs. 92–96) by manipulating the slidebars that control RBG, HSV, biases, and brightness and then click the Modify button under the drop-down listbox of prefab colors and it will change the definition of that color accordingly plus save it to disk. The next time you run the system, the new color definition will be loaded. Similarly, you can set the slidebar controls or move the cursor around the color wheel to obtain the shade you like, type a name into the New Color text box, and click Add. The new color will be added to the prefab color set and written to disk, which will be loaded the next time the system starts (Fig. 92).

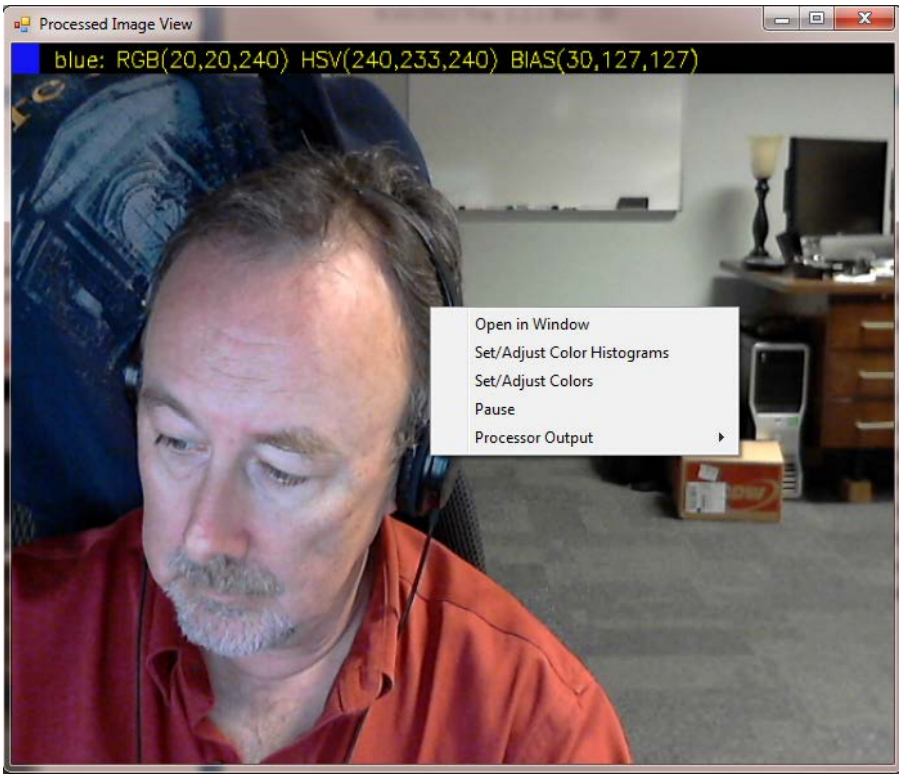


Fig. 92 Context menu showing color picker option

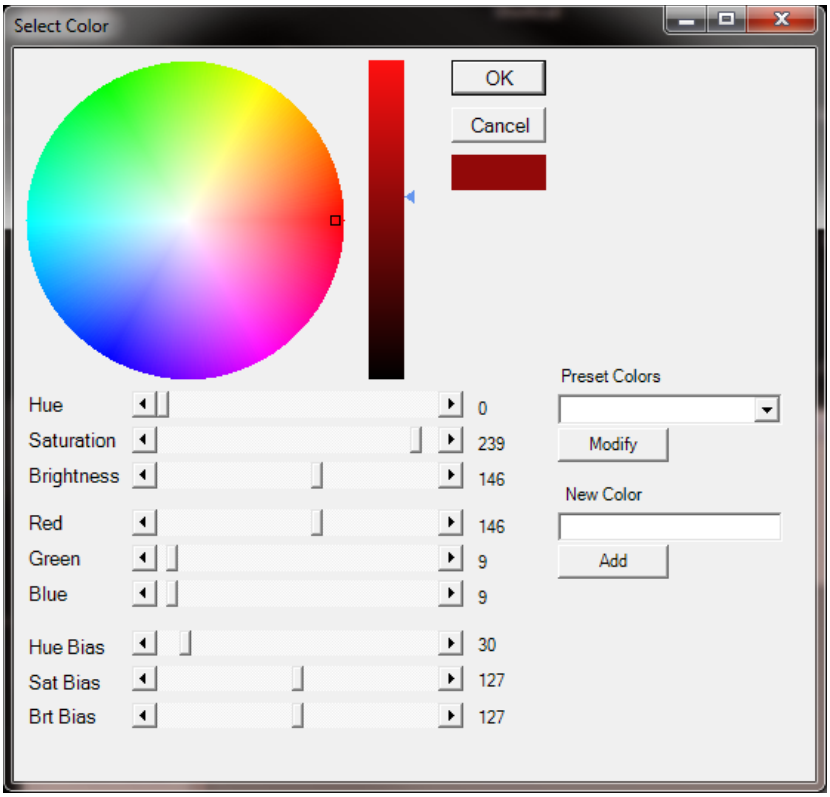


Fig. 93 Color picker

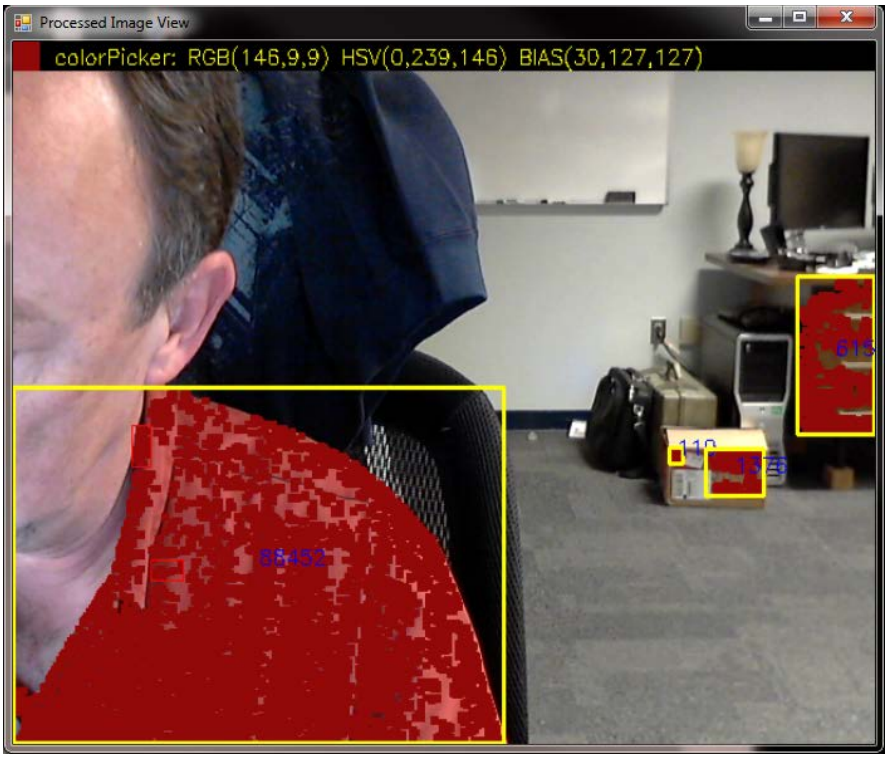


Fig. 94 Result of color picked from color picker

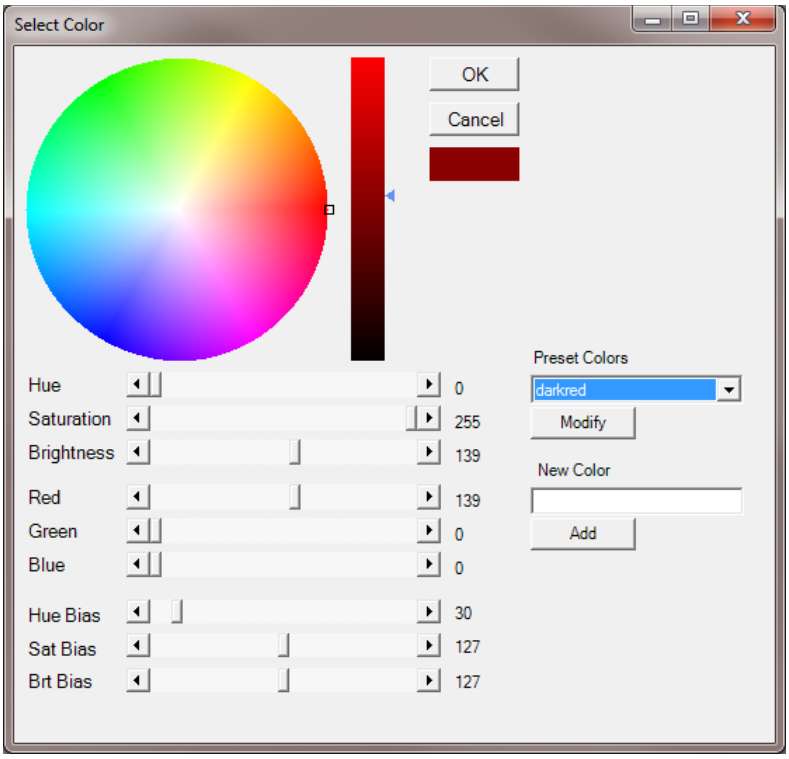


Fig. 95 Prefab color selected, dark red

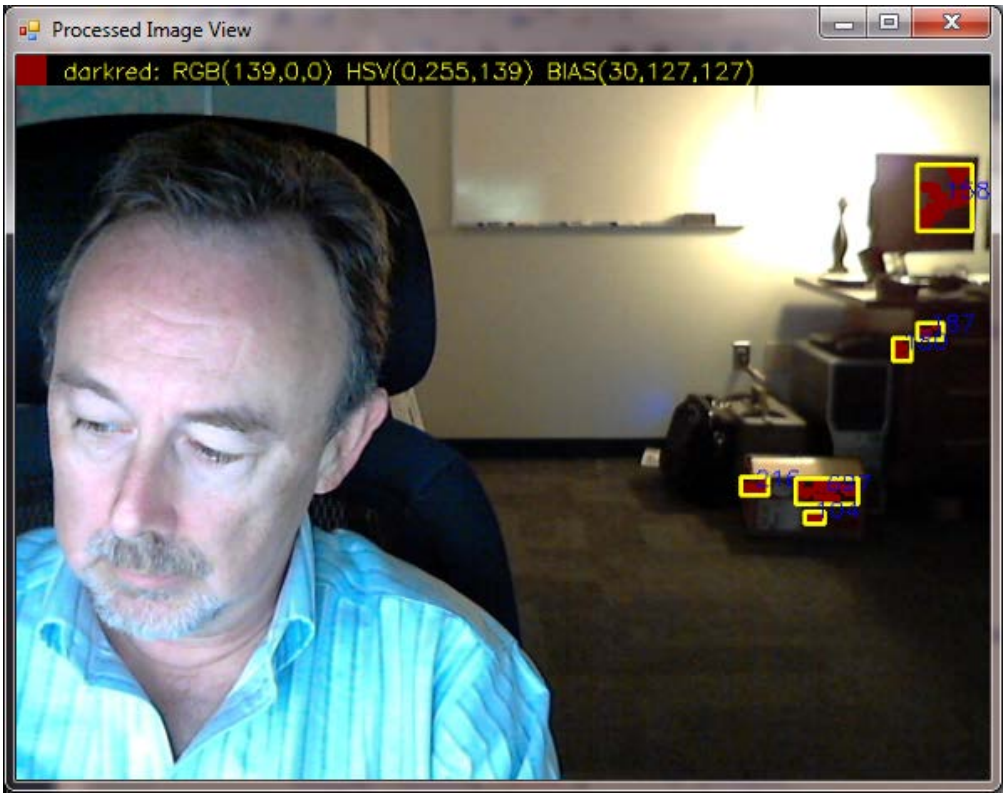


Fig. 96 Prefab color, dark red, being detected

Color Cycling

The color cycling function allows the operator to ask the robot to find colors in a particular range of a standard primary color like red, green, blue, or yellow. This can be accomplished by specifying in a goal or by setting property values on the Properties page. The operator robot will cycle through the range of colors specified and will return the largest region found in that range. Optionally, the region found can be sent to the PredatorTLD or Tracking subsims so that it can be followed.

7.23.2 Color Histogram Detection

The color HistogramChooser (Fig. 97) can be invoked by right-clicking on the processed image panel and selecting Set/Adjust Color Histograms. Use the mouse to set the ranges for the histogram by clicking either in or just below the Hue Spectrum. There are preset histograms available in the drop-down on the left in the center of the panel that give the general color and the hue range they employ. Selecting one of these and then pressing Test will engage the histogram detector. If you choose a preset color histogram and wish to adjust it, you can set the pointers to the range that is acceptable and then press Update. This will reset the definition of the color histogram to reflect the new hue ranges. If you wish to create a new histogram using this panel, enter the new color name in the New Colors field, set the pointers to the desired range, and press Add. The new color will be added to the table, and the next time you use the histogram panel it will be in the preset drop-down menu. The colors that appear initially in the drop-down are internally generated with arbitrary hue ranges and kept in facts. These facts can be saved and reloaded for future runs. If they are, they will be loaded but if not, the defaults are generated. An example of a color detection is shown in Fig. 98.

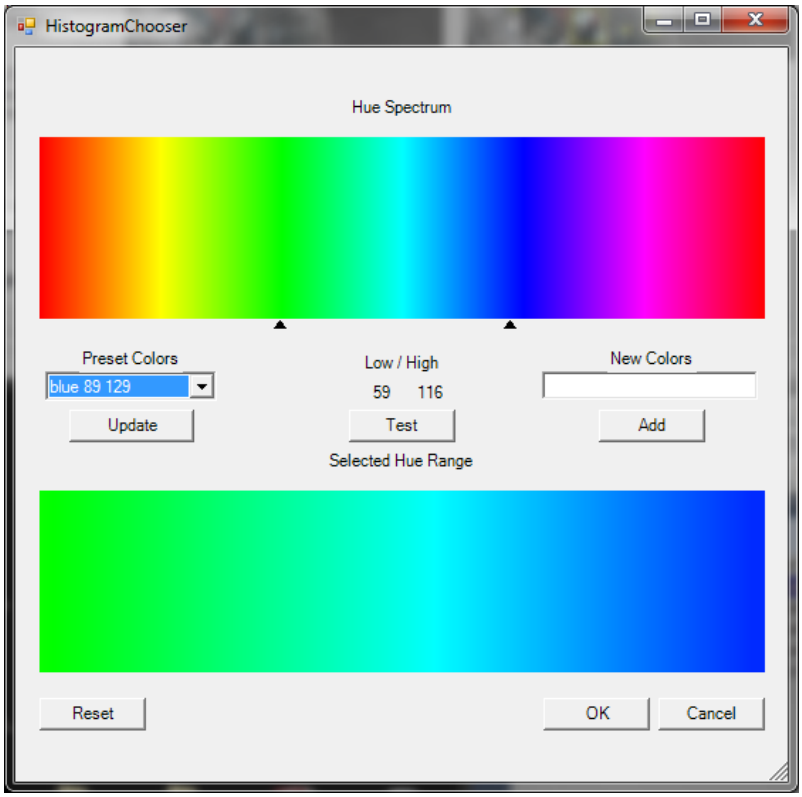


Fig. 97 Color HistogramChooser

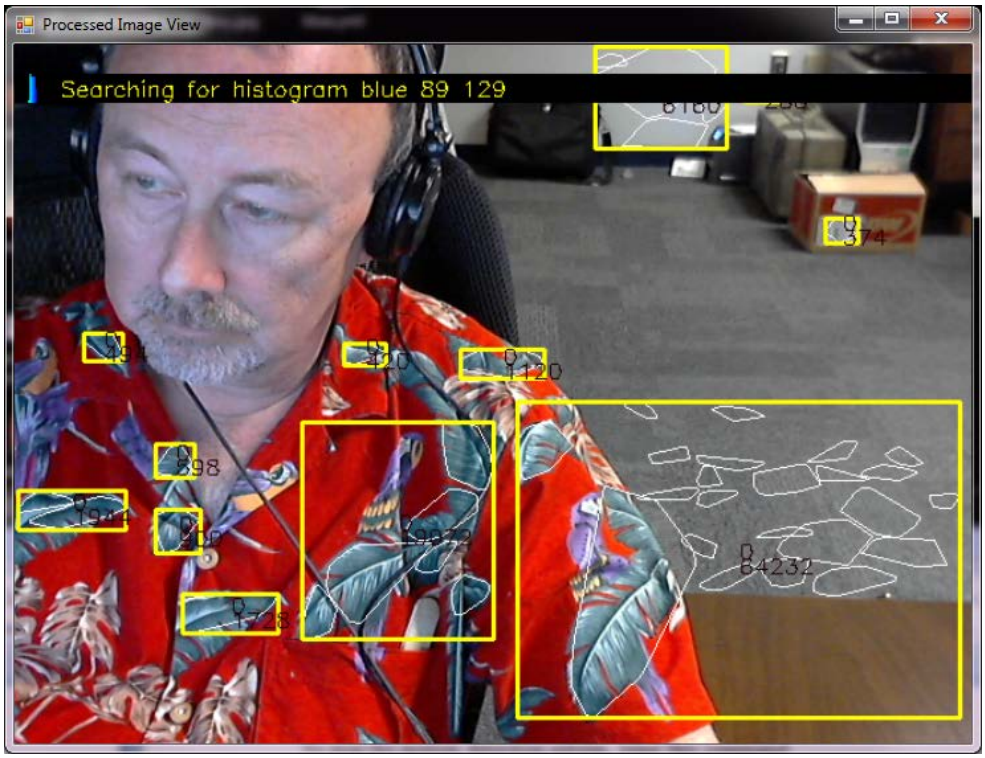


Fig. 98 Blue 89, 129 histogram detections

Note the status band near the top of the screenshot. At the left it depicts the histogram of the range being searched. You might have noticed the same status band at the very top of the screenshot for single-color detection. The subsim can do both simultaneously but the results are mixed together so it is recommended you run multiple instances of this subsim to separate the operations. Each subsim can search for a different color or histogram, and each will publish result facts that are unique to it. Also, the status lines will appear only if the DebugLevel is above Silent.

Commands

Execs (color savecolors)
This command saves the current fact set of defined colors (including any current user adjustments or additions) to the file colors/colorFile.txt, which was mentioned at the beginning of this section.

Execs (color pickcolor)
This command brings up a dialog to allow you to select color. Upon clicking OK, the subsim will start detecting the new color. This facility is useful for a quick pick. The findcolor command is a bit more versatile.

Execs (color setminarea)
This command allows the operator to set a minimum value for the area of a color region detection. Anything under this area (expressed in square pixels) will be rejected.

Execs (color setmaxarea)
This command allows the operator to set a maximum value for the area of a color region detection. Anything over this area (expressed in square pixels) will be rejected.

Execs (color startcolortracking)
This command starts single-color detection as previously described.

Execs (color stopcolortracking)
This command stops single color detection as previously described.

Execs (color findcolor red)
This command begins single-color tracking for the color called red. The subsim will examine each frame for this color, report the regions found that contained the color, and fit the area criteria. These values are updated after each frame and will continue to be until the color tracking changes or stops. Examples of facts reporting these areas (and similar areas found through histogram searches) and the results of this command follow the command descriptions. After each frame is processed, the

state variable corresponding to the current Colortracking subsim instance (in this case it is called color) will contain the name of the color.

Execs (color findcolor red)*
This command begins single-color tracking for the color called red and all of its variants (red0, red1, red2, etc.), which have been arbitrarily arranged based on the sorted hue/saturation/brightness values of the preset color assortment: red, green, blue, brown, black, and white. If autocycling is on (see next command: autocycleon), the subsim will examine each frame for this color and report the regions found that contained the color and fit the area criteria. These values are updated after each frame and will continue to be until the color tracking changes or stops. (Examples of facts reporting these areas and similar areas found through histogram searches and the results of this command follow the command descriptions.) If autocycling is not turned on, the subsim will await the command nextcolor to begin the detection with the first/next color in the list. When the autocycle is complete, the state variable corresponding to the current Colortracking subsim instance (in this case called color) will report done=true, and if it reports found=true, the region found will also be described: area, depth (if available), centroid x and y values, and the color it matches.

Execs (color autocycleon)
This command can be used in conjunction with the findcolor wildcard command as described. It can be called before or after the findcolor command. If before, it will begin cycling as soon as the findcolor command is issued, and if after, then commensurately it will begin afterwards. After the end of the cycle, the cycling will be turned off automatically and the results published in the state fact, which will be described following the command descriptions.

Execs (color autocycleoff)
This command can be used in conjunction with the findcolor wildcard command. It can be called after the findcolor command and will halt cycling (if it was on) at the current color in the list. Issuing the autocyleon command resumes the cycling. If you wish to manually move to the next color in the color list, then issue the following command:

Execs (color nextcolor)
This command can be used in conjunction with the findcolor wildcard command. It can be called after the findcolor command, and if autocycling is not active, it will move to the first/next color on the list.

Execs (color depthon)
This command has the effect of displaying the depth value (if available; if not, 0 is displayed) of a given region of color on the screen in its bounding box.

Execs (color depthoff)
This command has the effect of turning off the display of the depth value of a given region of color on the screen in its bounding box.

Execs (color areaon)
This command has the effect of displaying the area value of a given region of color on the screen in its bounding box.

Execs (color areaoff)
This command has the effect of turning off the display of the area value of a given region of color on the screen in its bounding box.

Execs (color stopcycle)
This command takes the subsim out of multicolor search mode. This mode is automatically turned on when the wildcard findcolor command is issued.

Execs (color capturehistogram painting gallery)
This command allows the user to select a portion of the image with the mouse (generate a bounding box) and create a histogram of that region's colors to store and to search. In this instance, the subject is painting, presumably a region containing a painting on a wall. The category (subdirectory or subfolder) it is to belong to is called gallery.

Execs (color findhisto painting gallery)
This command allows the user to search the image for regions of color matching the components of a given stored histogram. In this instance, the subject is called painting—a collection of the color content of a painting previously captured and stored—and the category (subdirectory or subfolder) from which it is to be loaded is called gallery.

Execs (color stopfind)
This command will instruct the subsim to cease searching for color regions that match the current histogram.

Execs (color ontrackclobs)
This command will instruct the subsim to send detected regions to the tracking subsim (Tracking or PredatorTLD or both depending on which recipients are specified in the config file). If the clobs (color blobs or regions of color) are sorted, they will be sent largest first. This will continue until the subsim is told to stop using the next command.

Execs (color offtrackclobs)
This command will instruct the subsim to cease sending clobs to the tracking subsim(s).

Execs (color userdefinehistogram red 0 14)
This command will instruct the subsim to create and store a histogram made up of red, whose hue values fall between 0 and 14. This will be saved to disk, published in the histogram facts, and reloaded upon next startup.

Execs (color generatedefaultcolorhistograms)
This command will instruct the subsim to create and store a set of histograms that comprise the major colors of the spectrum (red, orange, yellow, green, blue, indigo, magenta, and one with all the colors called FullMonty). Like the ones created with the userdefinehistogram command, these are stored in facts and on disk and reloaded upon next startup. If they do not exist upon next startup (due to some bizarre filing accident, maybe), they will be regenerated automatically and saved.

Published Facts

This subsim generates facts that report the current status of detections to the robot and contain information that can be used to select and follow targets and in general contribute to the eventual combination of sensor readings that will be used to identify objects in the FOV (Figs. 99–102).

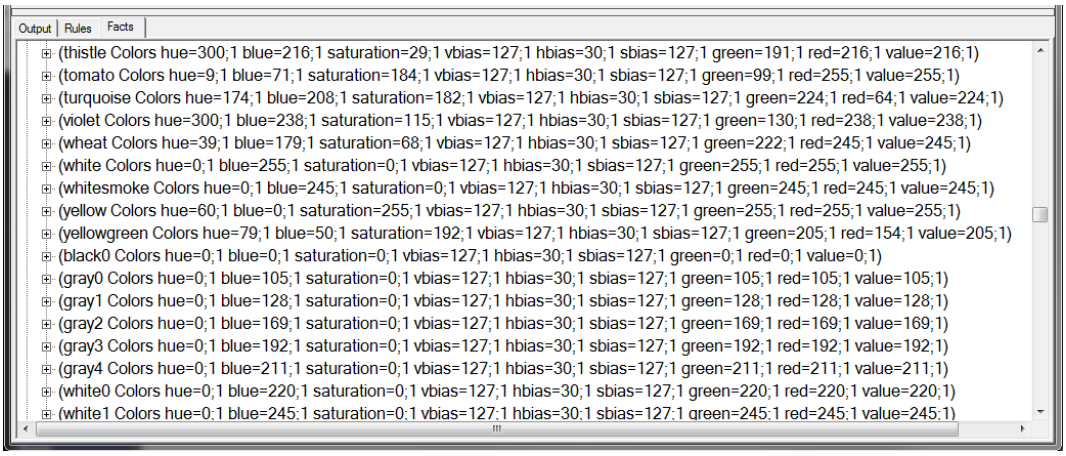


Fig. 99 Facts containing color parameters

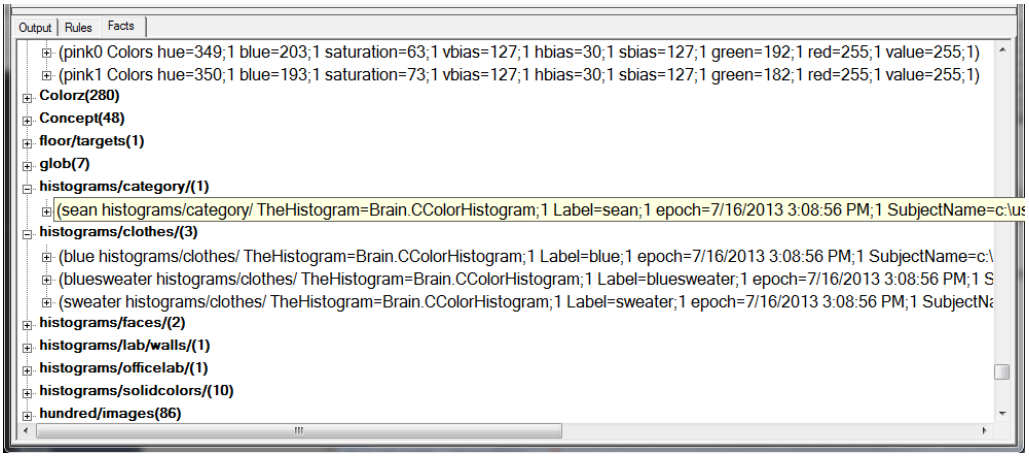


Fig. 100 Facts containing color histograms and their parameters

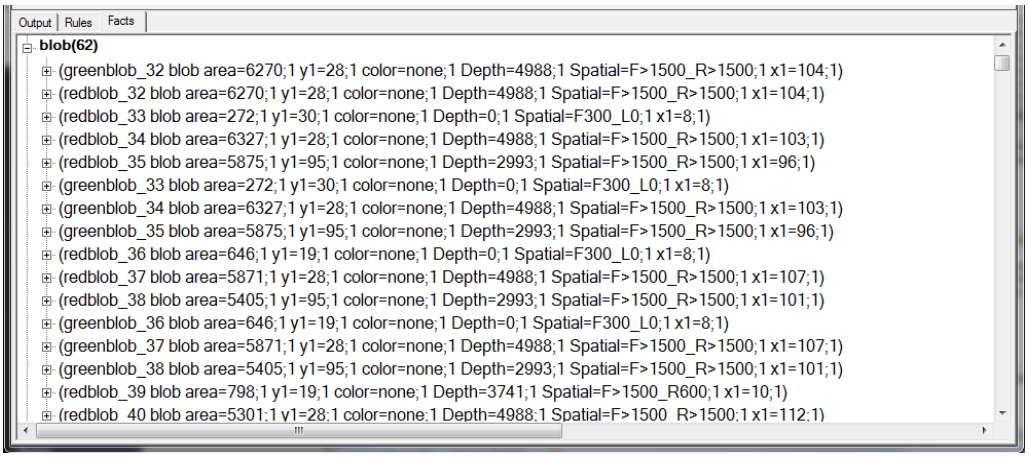


Fig. 101 Facts containing colorblob detections and their parameters

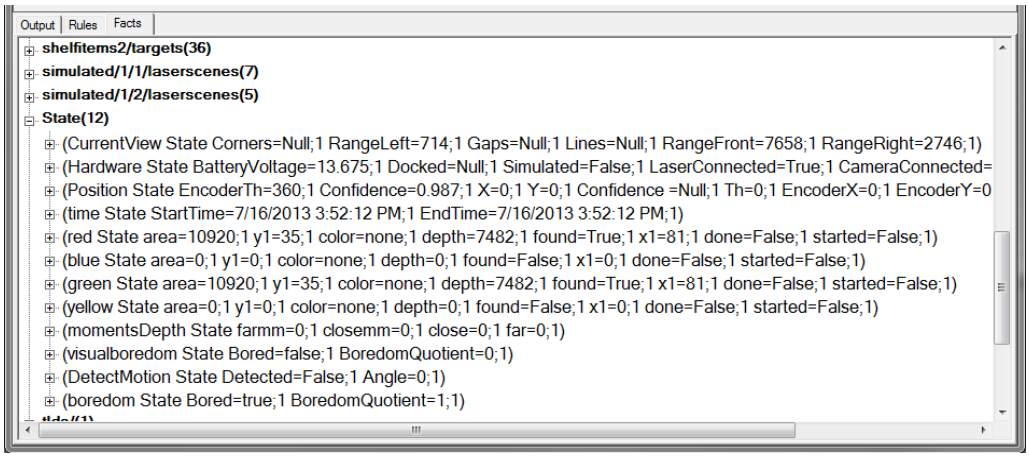


Fig. 102 State facts containing detection information red, green, blue, and yellow Colortracking instances

7.24 Motion Detection

The ability to detect movement within the robot's visual scene is quite useful. To that end, a Motion Detection algorithm (gleaned from an openCV demonstration) has been implemented in SS-RICS.

The output from Motion Detection consists of a fact of type State and name DetectMotion (which will report whether or not motion is detected and which direction it is heading) and a group of facts of type Moving whose names are the names assigned to the regions detected such as bogey_1 or bogey_223. In the Moving facts are various slots including the following:

- y1 and x1, which are the x,y coordinates of the centroid of the Moving region.
- Angle, which is the approximate direction the moving region is going. This is expressed as an angle in degrees from 0° to 360° where 0° and 360° point straight up, 90° points right, 180° points down, and 270° points left.
- Spatial contains (if available) a class instance of an object containing x,y,z 3-D positional from the robot's frame of reference.
- Depth contains (if available) the distance of the camera to the target or in this case whatever moved in the FOV that caused the detection.

Published Facts

This subsim generates facts which report the regions of motion detected (Fig. 103).

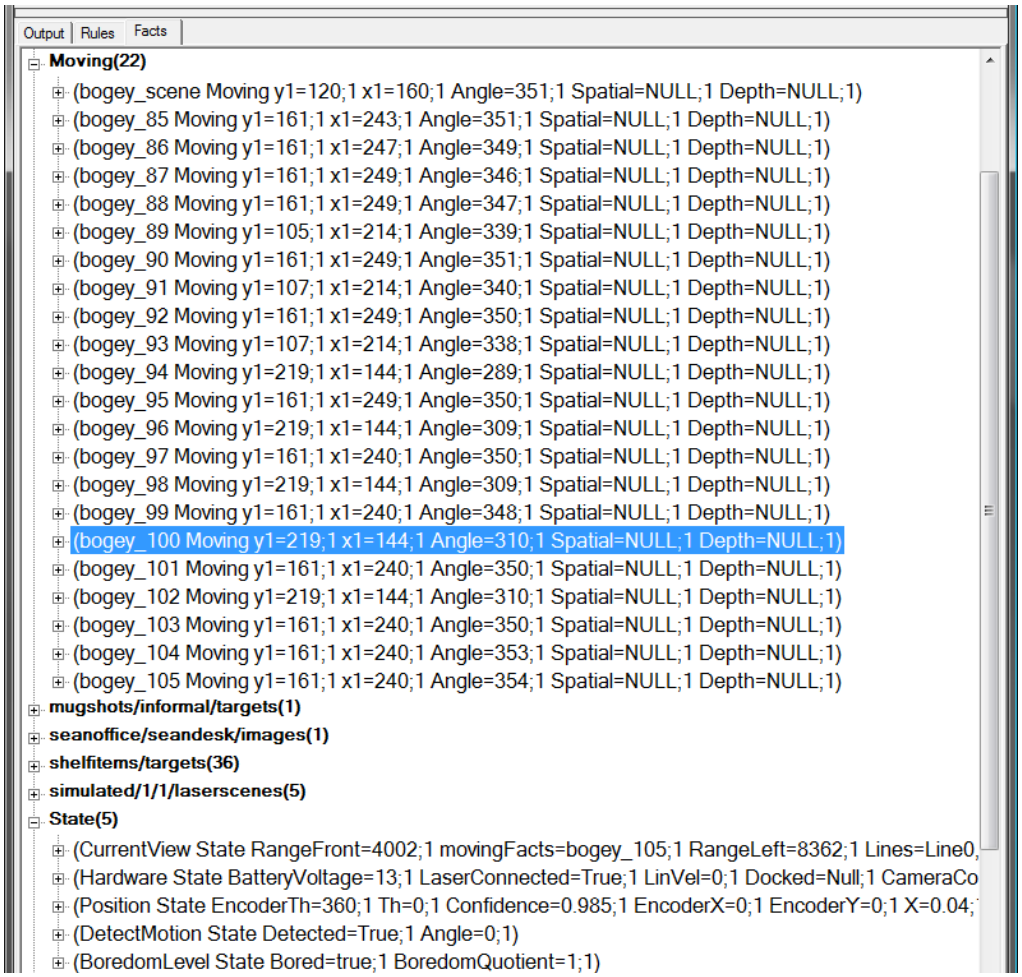


Fig. 103 Facts reporting motion detected

When the robot is paying attention to this fact, it can opt to follow the motion with its PTZ camera and its motion capabilities to the right or left. If a stereo camera is used, the depth data from each detection are included in the slots Spatial and Depth. If not, those fields are given a value of NULL. Additionally, the CurrentView State fact includes a slot called movingFacts that lists the facts generated by each motion detected (Moving). The actual pixels of the image are mapped onto the blue-screen scene differences view in a small rectangle with the centroid of motion as its center.

Figures 104–106 are examples of the 2 visual modes.

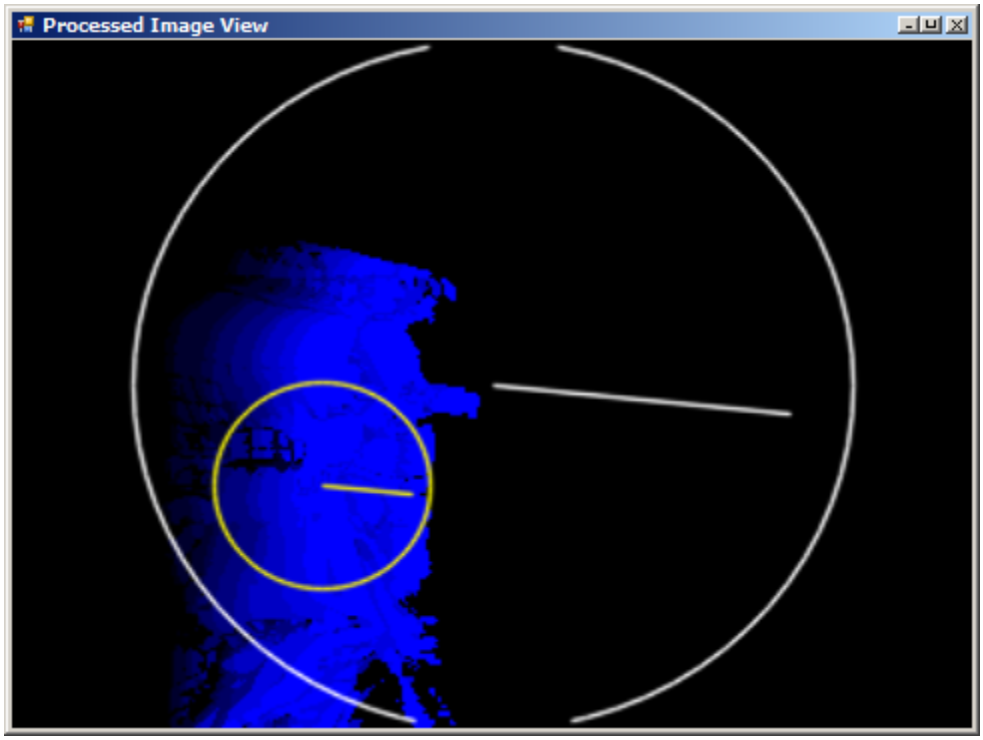


Fig. 104 Scene differences view

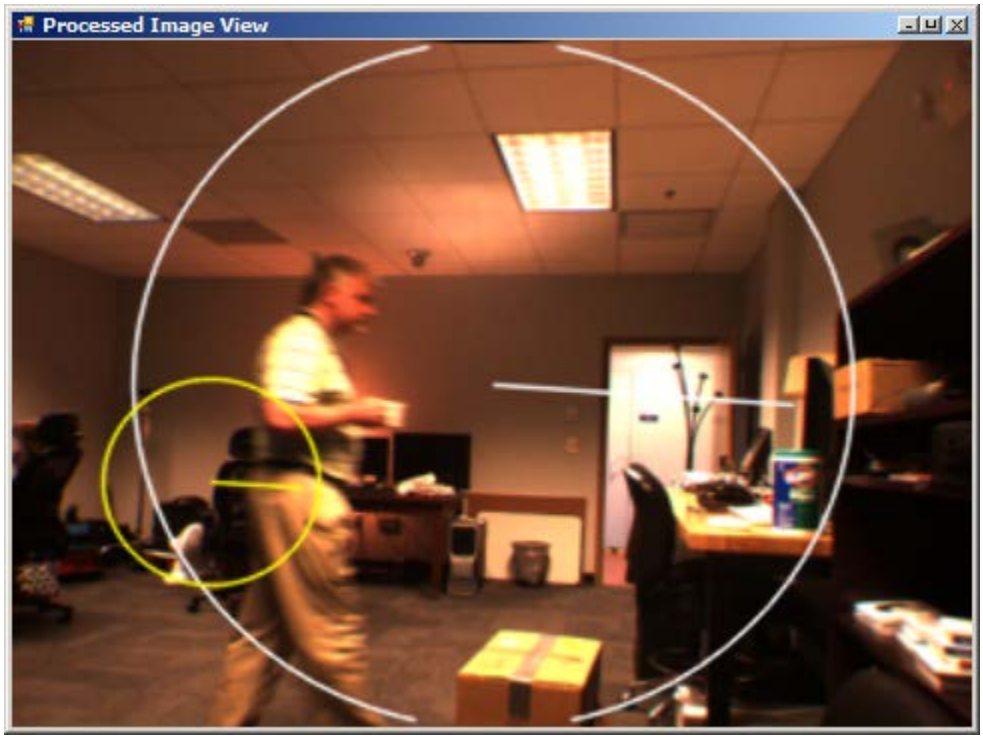
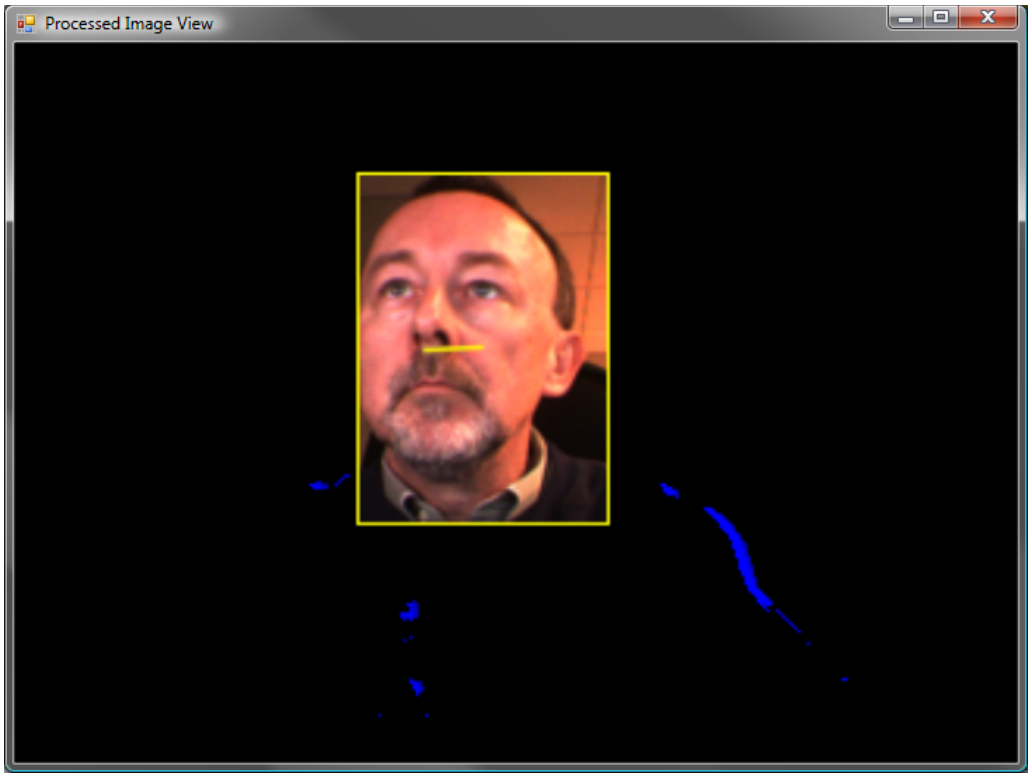


Fig. 105 Conventional view



Note: When the robot is moving, Motion Detection is disabled until the robot stops. Also, Motion Detection is "on" by default at startup.

Fig. 106 Image pixels mapped to scene differences

Commands

Execs (motiondetect startmotiondetect)
This command starts detecting motion.

Execs (motiondetect stopmotiondetect)
This command stops detecting motion.

Execs (motiondetect startmotiontracking)
This command instructs the Motiondetect subsim to send all detected motion to the Tracking subsim. It must be used in conjunction with either the singletrack or multitrack command so it knows which mode to be in.

Execs (motiondetect stopmotiontracking)
This command instructs the Motiondetect subsim to stop sending detected motion to the Tracking subsim.

Execs (motiondetect tracksingle)
This command instructs the Motiondetect subsim to send only the closest object detected provided depth data are available. If depth data are not available, the first motion detected is sent to the Tracking subsim.

Execs (motiondetect trackmulti)
This command instructs the Motiondetect subsim to send all of the motions detected to the Tracking subsim, which will then attempt to lock onto and track the regions sent to it, limited by the value of its maxtrackers property.

7.25 SceneDepth

The ability to find the average depth of a set of regions in the scene is provided with this subsim. The scene is divided into sectors inaccurately described as quadrants since there are 5 such regions with an additional sixth implied by combining the bottom and center portions of the screen with the depths displayed in Fig. 107. (Note that the front depth value is just below the center value.)

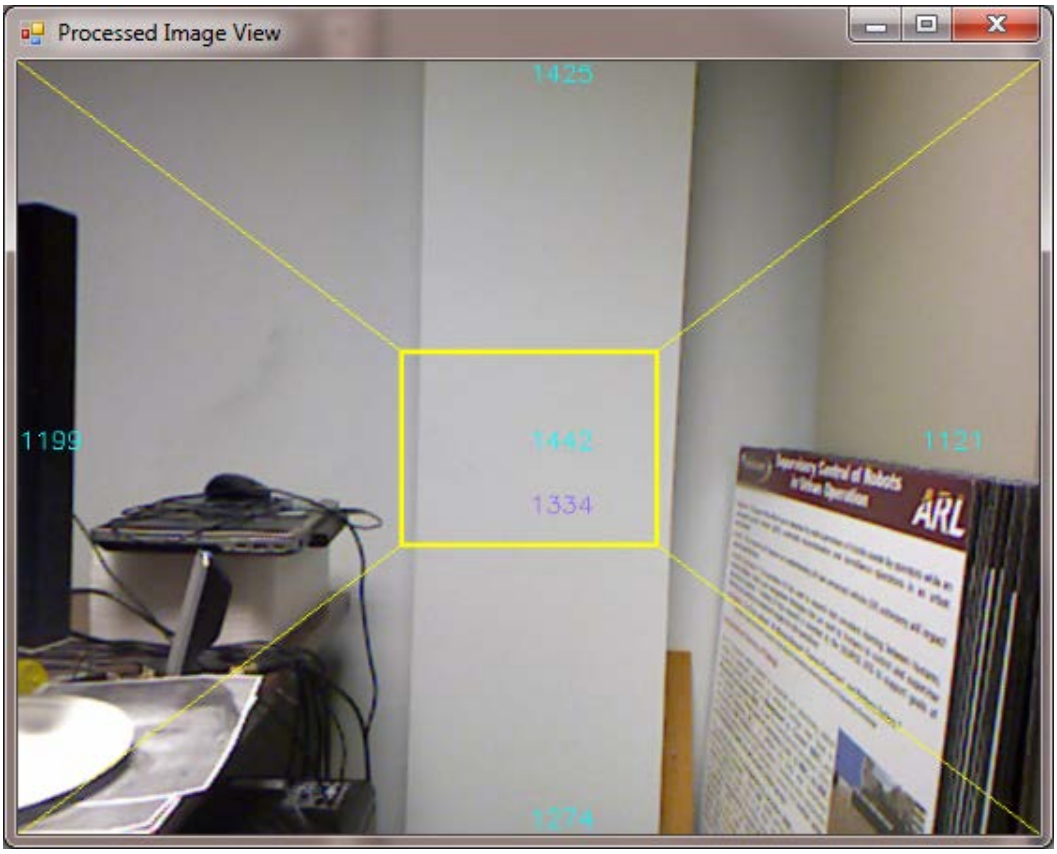


Fig. 107 SceneDepth display with depth values in millimeters

The depth values are published in a fact named SceneDepth of type State with the slots Top, Bottom, Left, Right, Center, and Front. Each contains the currently computed average depth of its respective sector. An additional slot, Profile, provides a string of 0s or 1s concatenated to represent whether or not a sector's average depth has exceeded a user-defined threshold. If the average depth is less than the threshold, the character "1" is used. If it is greater than or equal to the

threshold, the character “0” is used. They are arranged in the following order: Left, Top, Front, Center, Bottom, and Right. For instance, 000100 means that the Left, Top, Front, Bottom, and Right sectors are all at or over the threshold while the Center sector is under the threshold.

Commands

Execs(scenedepth setproximitythreshold <depth in millimeters>)
This command sets the proximity threshold in millimeters.

Execs(scenedepth computedepthcontinuously)
This command enables continous depth calculation and display.

Execs(scenedepth setcenterwidthandheight .25 .25)
This command sets the size of the center sector based on the specified ratio of the width and height of the image size. In this case, the center sector will be 1/4 of the height of the window tall and 1/4 of the width of the window wide.

7.26 Saliency

The ability to detect regions that can be regarded as “salient” or “important” within the robot’s visual scene is also quite useful. To that end, an algorithm that processes single frames and returns ROIs qualifying as salient has been implemented in SS-RICS. Saliency currently centers on edge detection and luminosity.

The output from Saliency consists of a group of facts of type Salient whose names are the names assigned to the regions detected such as salient1 or salient223. In the salient facts are the following slots:

- y1 and x1, which are the x,y coordinates of the centroid of the salient region.
- Spatial contains (if available) a class instance of an object containing x,y,z 3-D positional from the robot’s frame of reference.
- Depth contains (if available) the distance of the camera to the target or, in this case, whatever in the FOV caused the detection, which can be used to order the list of detections in the currentview state fact.
- Area gives the square pixel area of the bounding box containing the region detected, which can be used to order the list of detections in the currentview state fact.
- Saliency is the calculated value ranking the saliency, which can be used to order the list of detections in the currentview state fact.

Published Facts

This subsim generates facts that report the regions of saliency (Figs. 108 and 109).

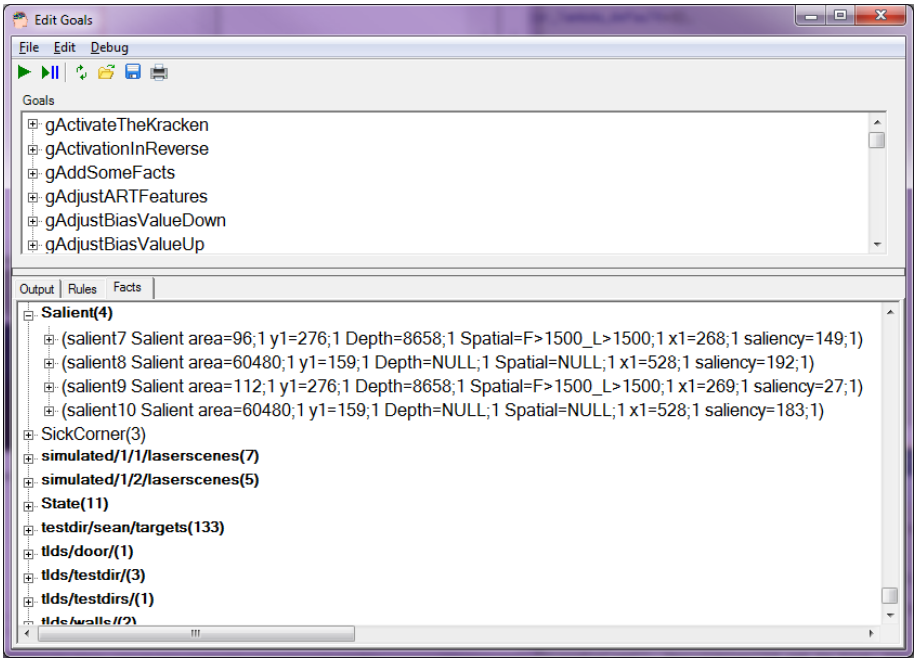


Fig. 108 Facts reporting detected salient regions

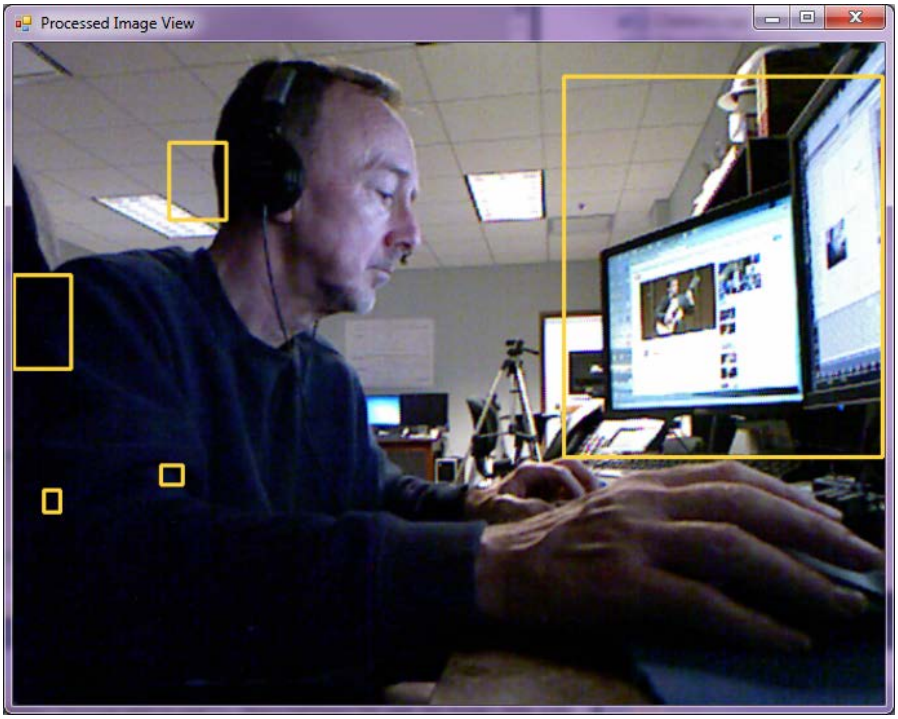


Fig. 109 Regions of saliency

When the robot is paying attention to this fact, it can opt to follow the motion with its PTZ camera and its motion capabilities to the right or left. If a stereo camera is used, the depth data from each detection are included in the slots Spatial and Depth. If not, those fields are given a value of NULL. Additionally, the CurrentView state fact includes a slot called Salients that lists the facts generated by each salient detected (Salient).

When the robot is paying attention to this fact, it can opt to follow the motion with its PTZ camera and its motion capabilities to the right or left. If a stereo camera is used, the depth data from each detection are included in the slots Spatial and Depth. If not, those fields are given a value of NULL. Additionally, the CurrentView state fact includes a slot called sSalients that lists the facts generated by each salient detected (Salient).

Additional Commands

Execs (saliency sortlistbyarea)
This command will sort the list of salients that is stored in the currentview state fact by area in descending order.

Execs (saliency sortlistbydepth)
This command will sort the list of salients that is stored in the currentview state fact by depth in descending order.

Execs (saliency sortlistbysaliency)
This command will sort the list of salients that is stored in the currentview state fact by saliency in descending order.

Execs (saliency sortlistbynone)
This command will present the list of salients that is stored in the currentview state fact by order of detection.

Execs (saliency setcapturelevel capture|track|trackandcapture)
This command instructs the subsim to tag detections for tracking, template capture, or both when emitting them to subscribing subsims.

Execs (saliency setscalefactor <real number up to 1.0>)
This command instructs the subsim to scale the input image by the value specified to speed up processing. A value of 0.5 results in reducing the image by half along both width and height dimensions, resulting in an image a quarter of the size of the original. This speeds up processing of that image at a cost of accuracy. The user can decide on the correct balance of speed and accuracy by trying various values and examining the results.

Execs (saliency startsendtargets)
This command instructs the subsim to emit targets to subscribing subsims.

Execs (saliency stopsendtargets)
This command instructs the subsim to halt emitting targets to subscribing subsims.

Execs (saliency settargetdir <directory name>)
This command instructs the subsim to set the directory name for storing targets sent to the template matcher. This will be a subdirectory of the targets directory.

Execs (saliency setdogmaxlevel <integer>)
This command instructs the subsim to set the saliency parameter dog_level to the specificed integer value. The default is 5.

Execs (saliency setscaledelta <integer>)
This command instructs the subsim to set the saliency parameter scale_delta to the specificed integer value. The default is 2.

Execs (saliency setgaborlambda <3 real numbers>)
This command instructs the subsim to set the saliency parameter gabor_lambda to the specificed trio of real values. The defaults are 12.5, 6.25, and 3.123 or 1/0.08, 1/0.16, and 1/0.32.

Execs (saliency setnangles <integer>)
This command instructs the subsim to set the saliency parameter nangles to the specificed integer value. The default is 4.

Execs (saliency setwincenter <real>)
This command instructs the subsim to set the saliency parameter win_center to the specificed real value. The default is 0.1.

Execs (saliency setwinsurround <real>)
This command instructs the subsim to set the saliency parameter win_surround to the specificed real value. The default is 0.6.

Execs (saliency setsubsample <integer>)
This command instructs the subsim to set the saliency parameter sub_samples to the specificed integer value. The default is 1.0.

Execs (saliency setpriorpixels <real>)
This command instructs the subsim to set the saliency parameter prior_pixels to the specificed real value. The default is 1.0.

Execs (saliency resetdefaults)
This command instructs the subsim to reset the above described 8 parameters to their original default values.

7.27 Segmentation

The ability to take detected regions (primarily emitted by the Saliency subsim, but which can come from any visual subsim that can emit a bounded region) of the screen and attempt to recognize them based on shape is a natural companion to the Saliency subsim.

The output from the Segmentation subsim consists of a group of facts of type segment whose names are the names assigned to the regions detected such as segment1 or segment223. Classifications are obtained from trained ARTMap neural networks. In the segment facts are the following slots:

- Osclassification: classification that comes from so-called "Superquadratics" and classifies based on shape alone.
- Artclassification: classification which comes from a Histogram of Oriented Gradients (HOG) features.
- Artosclassification: combined classification from both Shape Only and HOG features.
- Geon: general shape that fits the object being segmented. Possible geons include Cylinder, Ellipsoid, and Cuboid.
- Quad: specifies which sector of the screen it appears in: left, right, center, top, and bottom.

The ARTMap neural network files (trainable) and the yaml files (pretrained) to facilitate segmentation, learning, and classification, are kept in a subfolder of the robot's working directory.

Published Facts

This subsim generates facts that report the segments (Figs. 110 and 111).

Fig. 110 Facts reporting detected segments

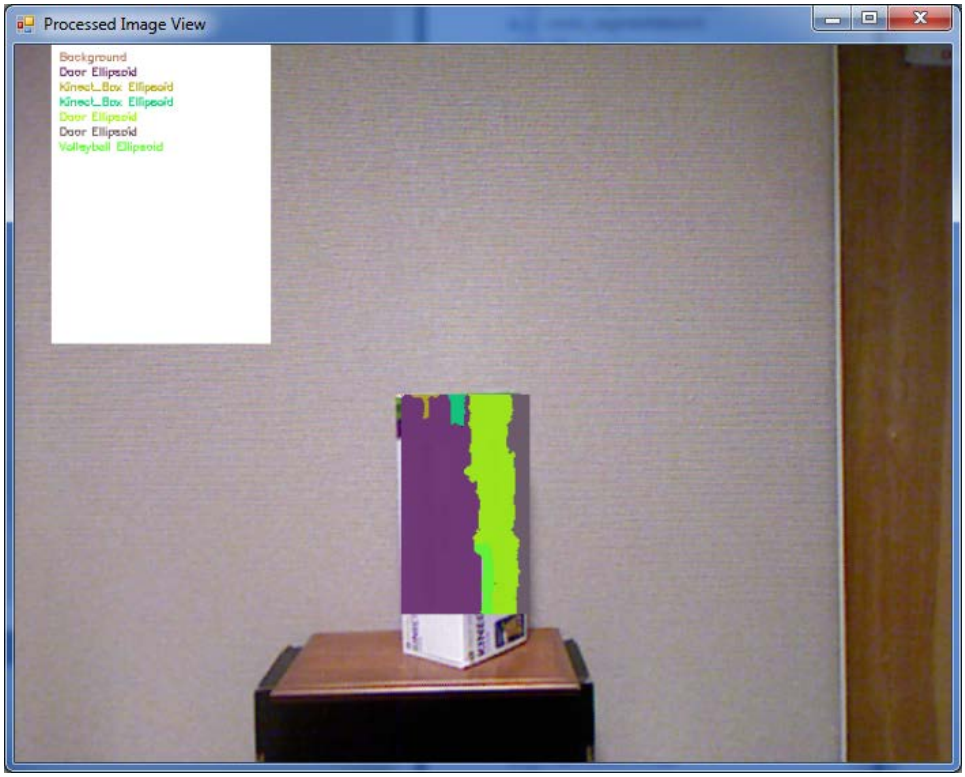


Fig. 111 Segmented ROI

Config File

The cors.cfg file has to be located in the same directory as the executable. The parameters in the file that might be modified on a regular basis are listed in Table 3. Apart from the parameters listed in the table, there are module specific parameters that are not explained in detail in this manual.

Table 3 Configuration file parameters in cors.cfg

Parameter	Description
[Interface]input directory	Directory for previously stored data used by Play.
[Interface] output directory	Directory for storing new data from Kinect.
[Interface] objects	Comma-separated list of objects for recognition. This list order is important and cannot be changed. New objects can be added at the end of the list.
[Interface] depth threshold	The distance threshold in meters beyond which point cloud data are not processed.
[Interface]_use rgbd constants	Uses the Kinect camera parameters for the rgbd data set. Should be false for normal operation.
[Detector] classifier file	Location of the Support Vector Machine (SVM)-based classifier using both Superquadrics and HOG features.
[Detector] classifier file only shape	Location of the SVM-based classifier using only the SQ features.
[Detector] train directory	Location where the exemplars are stored when Save Object is clicked. Training uses this directory to train new classifier.
[Detector] kernel	Kernel type: RBF (Default OpenCV Kernel), or Custom kernel (for superquadrics matching).
[Detector] use_conic_segmentation	True to use conic segmentation algorithm and false to use NCUT segmentation algorithm.
[GeonDetector] svm file	Location of the SVM classifier used to detect geons.
[ArtClassifier] vigilanceT	Training vigilance parameter of ART.
[ArtClassifier] vigilanceC	Classification igilance parameter of ART.
[ArtClassifier] location	Location and filename of the ART classifiers. No file extension.

Additional Commands

Execs (segment initializeartmap)

This command will delete the existing ARTMap neural network files, allowing them to be recreated and trained anew.

Execs (segment clearscreen)

This command will clear the screen of the previous action's results.

Execs (segment setcenterwidthandheight .25 .25)

This command will set the size of the center sector of the screen. For several visual subsims, the screen is divided into "quadrants" (slightly erroneous given the center sector) called Left, Right, Top, Bottom, and Center. The values given in the

command are the ratios of the screen's width and height applied to the size of the center sector.

Execs (segment dosegmentation)
This command will initiate segmentation on the currently active ROI (commonly provided by the Saliency subsim but can come from any that emit an ROI) in the Segmentation screen. The segment data are input to the ARTMap neural net,which does a best fit on it and reports which classification it has computed (this value is an integer that represents the category which was used to denote the input training data for it). The results will be published in facts and on the screen. If there is no valid ROI, the command is ignored.

Execs (segment dotraining 3 box)
This command will initiate segmentation on the currently active ROI. Then the segments contained within the ROI (as computed by the Segmentation function) will be used to train the ARTMap to recognize those segments as box. The 3 refers to the supervisory number, or category, that the ARTMap uses for training and for classifying.

This subsim and the Saliency subsim currently require a Kinect camera using the Freenect openKinect software drivers. It is planned to expand their functionality to allow for any stereo camera in the future.

7.28 PredatorTLD Tracker

The ability to acquire and track an object in the visual field (such as a face detected by the Faces subsim) is very important to the cognitive facilities being implemented in SS-RICS. The ability to lock onto, follow, detect, and learn an object or target is crucial to this effort. To make this happen we use tracking software developed at the University of Surrey (United Kingdom) that is the brainchild of Dr Zdenek Kalal (2010), whose specialty is the visual perception capabilities of computers. We have implemented his algorithm, which takes a 3-fold approach to this problem:

1. A region of the image is bounded by a box and features of interest in the box. They are tracked using the Lucas-Kanade algorithm implemented in the openCV function cvCalcOpticalFlowPyrLK.

2. The actual appearance of the target is learned using 15-pixel-square snippets of the object itself (positive example) and images surrounding the object chosen randomly (negative examples). As tracking continues, the appearance of the object is learned and aids in the detection of the object.

3. A Fern Classifier uses a tree structure to filter out false positive detections.

This 3-pronged approach is the basis for the PredatorTLD algorithm.

There are 2 methods to being tracking. First, to manually select a target, switch the processed view scene to predatortld by right-clicking on the image and selecting it from the drop-down menu (Fig. 112).

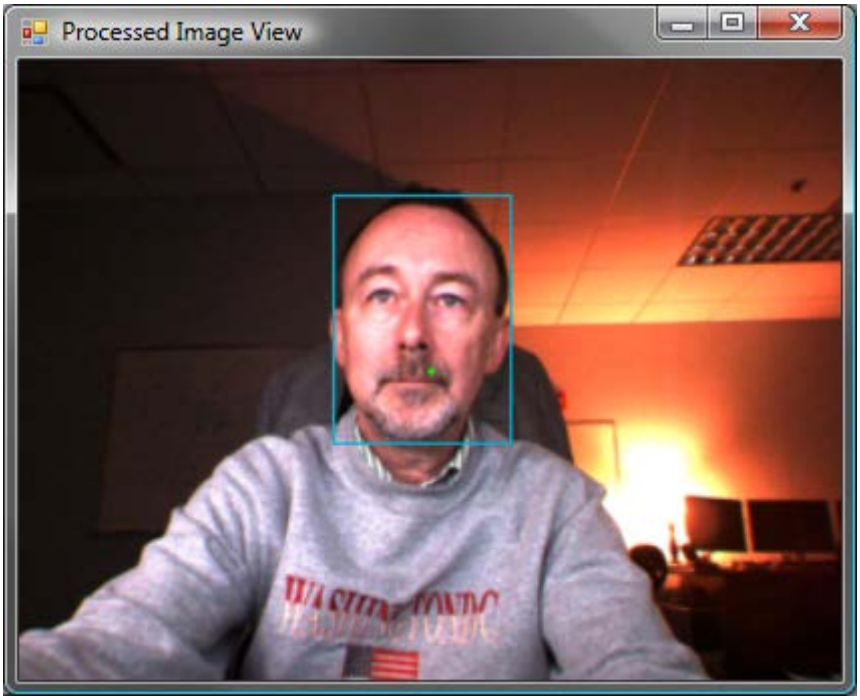


Fig. 112 Draw a bounding box around the target

Then issue the following command in a goal:

Execs (predatortld startpredator)
This command starts an instance of the predator tracker, whose output to the screen resembles Fig. 113. The current frame count and the confidence value the tracker computes from frame to frame of the target are displayed at top left.

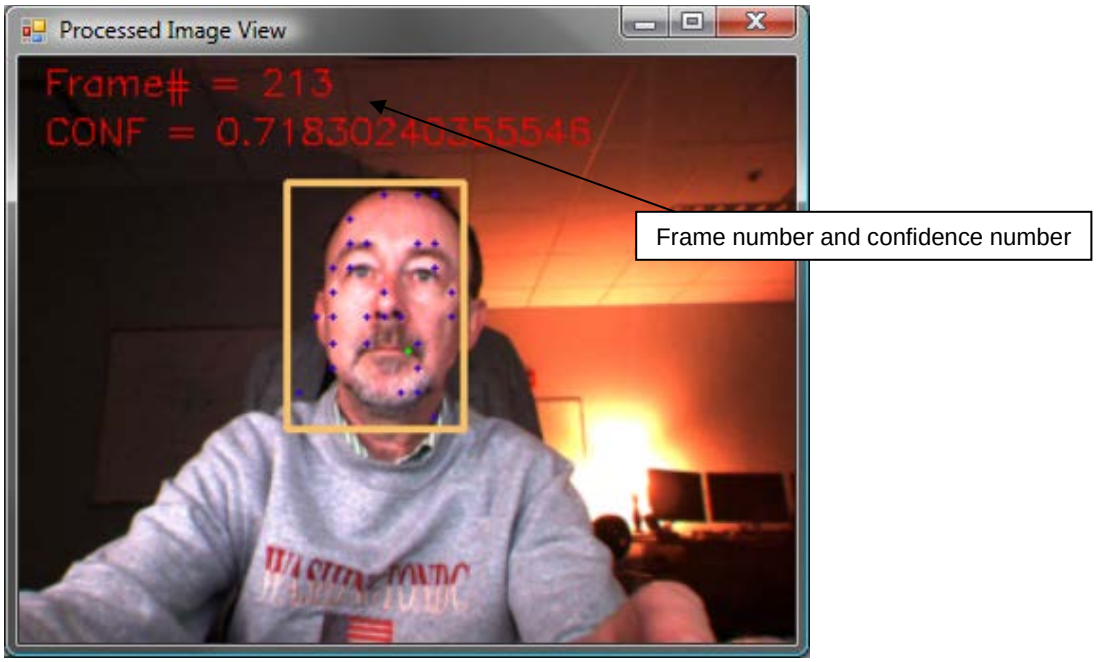


Fig. 113 Nominal tracking view

Figure 113 shows the standard marker for denoting the current position of the target. Two other markers are available: Convexhull and Crosshairs (Figs. 114 and 115, respectively). To select them, use the Properties menu for this subsim available in the subsim config dialog box.

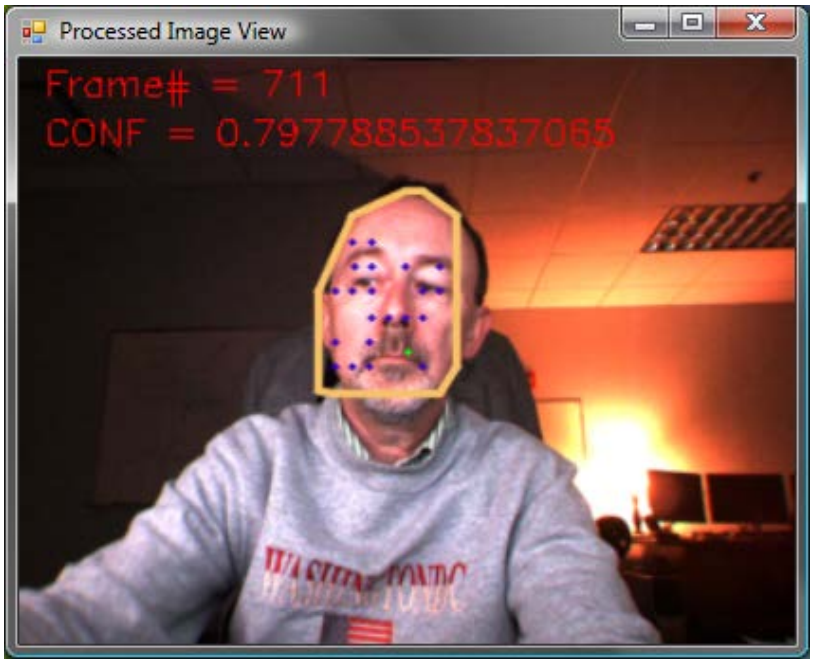


Fig. 114 Convexhull marker type

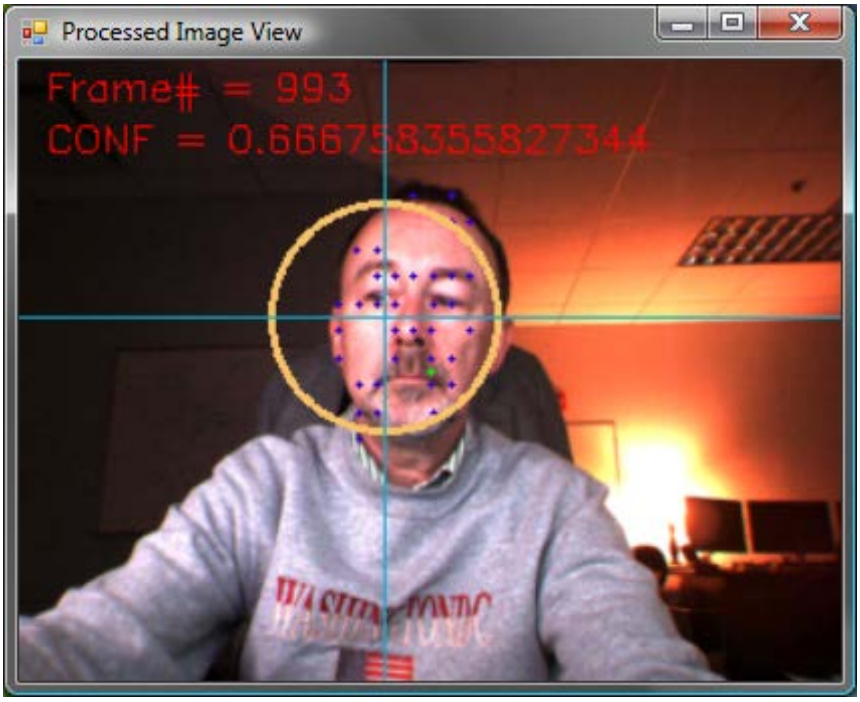


Fig. 115 Crosshairs marker type

Other commands that can be issued in a goal include the following:

Execs (predatortld stoppredator)
This command stops the instance of when the predator tracker started with the startpredator directive.

Execs (predatortld export workbot)
This command serializes the instance of a tracker and writes it out to disc (as a file named workbot.tld) for later reloading. Like the ImageCorrelation and TemplateMatch subsims, the name of the subject can include directory information such as tlds/bots/workbot, which will direct the export function to create (if necessary) a directory called tlds/bots in the current working directory of the program and store the object under the name workbot.tld.

Execs (predatortld import workbot)
This command deserializes the instance of a tracker from the file specified by workbot.tld and resumes tracking the object when/if it reappears in the FOV. Similar to the export function, the file name can be a path like tlds/bots/workbot, which tells the subsim to load the file workbot.tld from the directory tlds/bots.

Execs (predatortld togglepoints)
This command alternately turns on and off the display of feature points within the bounded area.

Execs (predatortld toggletraining)
This command alternately turns on and off the training function of the predatortld tracker. Normally, a confidence level of 0.7 or better is required for this function to activate.

Another way to track a face, as was the case described as manually selecting a target, is to let the Face Detection subsim send a candidate face to the PredatorTLD subsim via the input mechanism used by the Points and Line subsims to build representations of wall and gaps with the laser. When specifying which subsims to start up in the Config file, a version of the following line should be used:

```
<add name="predatortld" type="Brain.CSubSimProcessorPredatorTLD"
assembly="Brain" run="true" input="faces">
```

This tells the Brain that the Faces subsim will, when directed, send a candidate face to the predatortld subsim. The PredatorTLD subsim will launch a detector with the bounding box received from the Faces subsim and begin tracking it.

Another subsim processor called Tracking is more generalized and more flexible and can be fed face detections in the same way as is the PredatorTLD subsim. The PredatorTLD subsim is primarily used as a training tool. One picks an object and lets the tracker learn it and then export it for later detection using the Tracking subsim, which can load an arbitrary number of targets to detect. (See Section 7.29 for more details.)

The extension of the file .tld is the suffix convention adopted for this file type used in SS-RICS. Also, the PredatorTLD tracking software is quite robust and flexible but can fail to reacquire or lose its lock due to the viewing angle of the object or if the ambient lighting changes significantly. This is the bane of all computer-based visual-recognition systems. The ability to force the tracker to train can be used judiciously to teach it the object from multiple viewing angles and in varying lighting. While still not perfect, this technique can improve tracking and detection to a noticeable degree.

7.29 General PredatorTLD Tracking

This subsim is the successor to the one described in the previous section (PredatorTLD tracking). This is a more general-purpose system that accepts inputs from at least 3 different sources: the Faces and Motiondetect subsims and imported target files.

When specifying which subsims to start up in the Config file, a version of the following line should be used:

```
<add name="tracking" type="Brain.CSubSimProcessorTracking"
assembly="Brain" run="true" input="faces,motiondetect">
```

Tracking Facts

Upon subsim initialization, all tld subdirectories (tlds/, etc.) are scanned, and a fact describing each is created and displayed in the Facts tab of the Goal Edit dialog box. Figure 116 shows some examples.

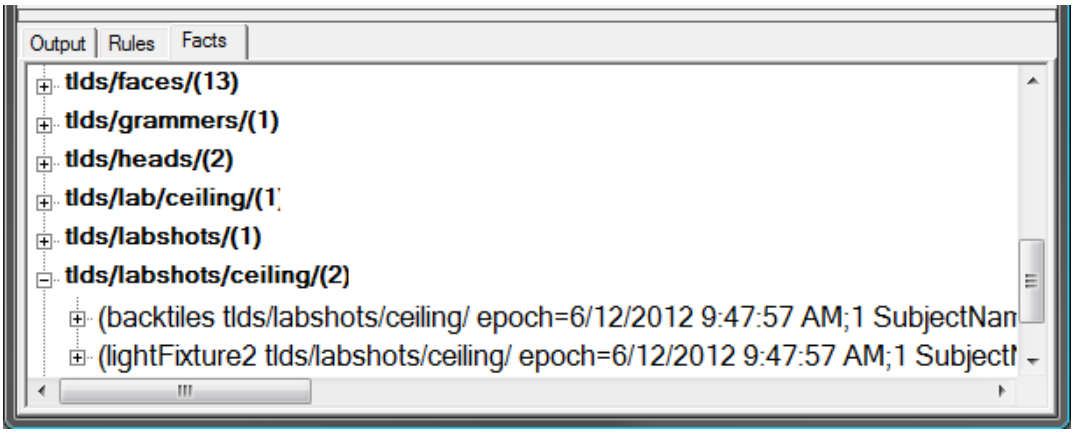


Fig. 116 TLD facts in the Facts window

The type of the fact is the name of the subdirectory containing the tld file. The fact itself contains the following fields:

- Name: this is the name of the file of the image minus the .tld extension.
- Type: as previoiusly described.
- Epoch: DateTime object string format of the name (this is the time that the image was saved).
- Label: same as Name.
- SubjectName: absolute full path name of the .tld file for the subject.

Commands

The commands executed when Export, Hibernate, shutdown, and WakeUp are set using this facility will operate on all trackers. They can be tailored to specify individual trackers and not just all trackers when executed through a goal. The following commands can be issued in productions (goals).

Execs (tracking loadtargets SEAN ERIC TROY)
This command tells the Tracking subsim to load the files SEAN.tld, ERIC.tld, and TROY.tld (in this case, target files of the faces of these 3 people) and begin tracking.

If no targets are specified, the command is ignored. If the target resides in a directory other than the current working directory, specify its full path (relative to the current working directory) in the command; for example, tlds/bogeys/SEAN instead of SEAN. This says a target is stored in the directory tlds/bogeys with the name SEAN.tld. If you wish to load all targets in a particular subdirectory, use the format: tlds/bogeys/*. (See the following export and rename functions for more insight.)

Execs (tracking shutdown SEAN ERIC TROY)
This command tells the tracking subsim to shut down the targets SEAN, ERIC, and TROY. If no targets are specified, all targets are shut down.

Execs (tracking hibernate SEAN ERIC TROY)
This command tells the Tracking subsim to "sleep" the targets SEAN, ERIC, and TROY (in this case, target files of the faces of these 3 people) and hide their tracking markers.

If no targets are specified, all targets are hibernated.

Execs (tracking wakeup SEAN ERIC TROY)
This command tells the Tracking subsim to "awaken" the targets SEAN, ERIC, and TROY (in this case, target files of the faces of these 3 people) and resume tracking. If no targets are specified, all hibernating targets are reactivated.

Execs (tracking rename bogey_443 SEAN)
This command tells the tracking subsim to rename the target nominally named bogey_443 (the name of the 443rd region of movement found by the Motion Detection subsim and handed off to the Tracking subsim) to SEAN. This name will then replace the old one on the screen and be used to name the file the tracker is stored in should it be exported. The target can also be renamed to include a destination path other than the current working directory so that the target can be saved in a specific directory. This could be tlds/bogeys/SEAN to specify its eventual export destination.

Execs (tracking export SEAN ERIC TROY)
This command tells the Tracking subsim to serialize the tracking objects for the targets SEAN, ERIC, and TROY and save them as files (SEAN.tld, ERIC.tld, and TROY.tld). These files can later be reloaded (using the loadtargets command, for instance) and deserialized to reinstantiate their tracking objects and redetect the

object they previous learned. If no targets are specified, all targets are saved. Similarly, if the rename example was bogey_443 and renamed tlds/bogeys/SEAN, then tlds/bogeys/SEAN could be specified in the list of the export command and the target would be save as a file named SEAN.tld in a directory called tlds/bogeys, which can be found in the current working directory.

Execs (tracking exportall <directory> <shutdown>)
This command tells the Tracking subsim to serialize all current tracking objects to the directory specified and deleted if the shutdown parameter is "true". If it is "false", the trackers will remain active.

Execs (tracking forcetrain SEAN ERIC TROY)
This command tells the Tracking subsim to force the tracking objects for the targets SEAN, ERIC, and TROY to go into learning mode. This overrides the confidence threshold of 0.7 that governs when trackers learn. This can useful as the tracker itself is rather conservative as to when it actively learns an object. If no targets are specified, all trackers are placed in this mode.

Execs (tracking ceasetrain SEAN ERIC TROY)
This command tells the Tracking subsim to tell the tracking objects for the targets SEAN, ERIC, and TROY to resume normal learning mode. This allows the trackers to learn only after the threshold of 0.7 is reached. If no targets are specified, all trackers are returned to normal learning mode.

Execs (tracking showpoints)
This command tells the Tracking subsim to display the features (points within the bounding box) being tracked on the screen. This command applies to all active trackers.

Execs (tracking hidepoints)
This command tells the Tracking subsim to hide (not display) the features (points within the bounding box) being tracked on the screen. This command applies to all active trackers.

Execs (tracking maxtrackers 5)
This command tells the Tracking subsim to limit the number of tracker instances to the number specified. The tracker's footprint is about 10 MB, so this is a good way of controlling its appetite.

Execs (tracking selectdisplaytype convexhull)
This command tells the Tracking subsim which type of marker to use to show where the object being tracked is in the FOV. The 3 types are Crosshairs (traditional target designator), Square (rectangle or bounding box), and Convexhull (similar to

crime-scene tape around a perimeter, it essentially ropes in all of the points). Section 7.28 gives visual examples of each of these.

Execs (tracking deletetlds sean tlds/faces/)
This command allows the user to delete a tld file and the fact that is generated for it. The parameter sean is the name of the file (implicit is the .tld suffix) and tlds/faces/ is the directory it is contained in. If the name sean is replaced with "*", all .tld files will be deleted from the directory along with their facts.

Figures 117–119 show some example scenes.

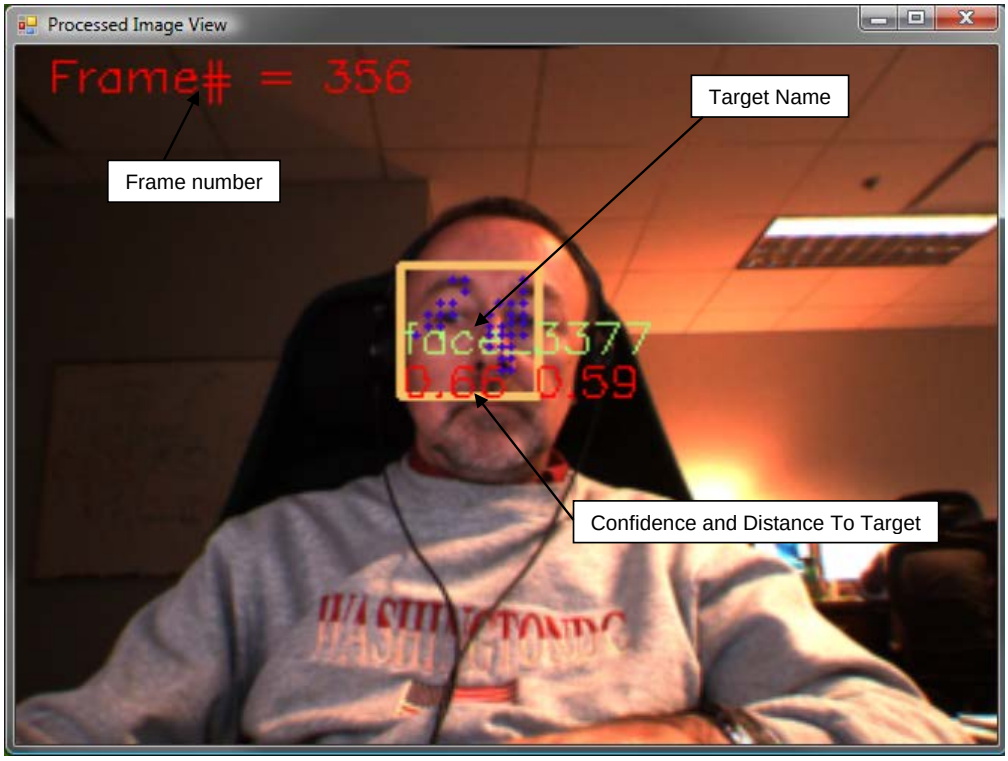


Fig. 117 Face sent to Tracking subsim from Faces subsim

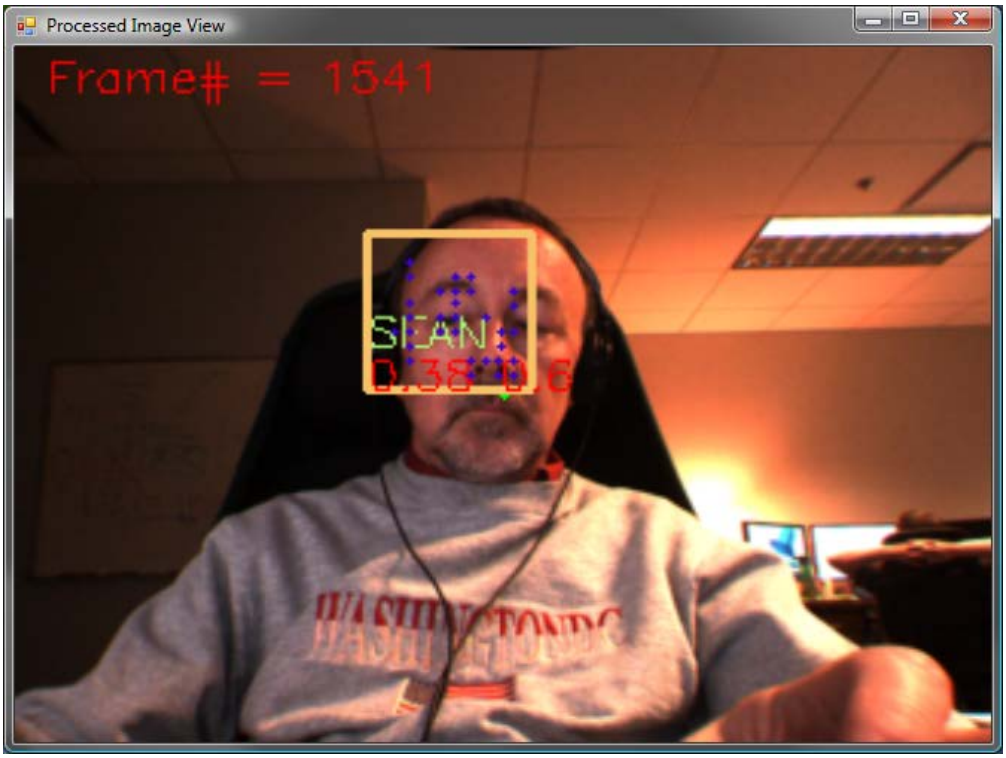


Fig. 118 Same face renamed from face_3377 to SEAN

Fig. 119 Same face with tracking features hidden

8. Activation Property Page

All memories (facts, goals, rules, etc.) have properties that control their existence and influence in the robots mind. An example is shown in Fig. 120.

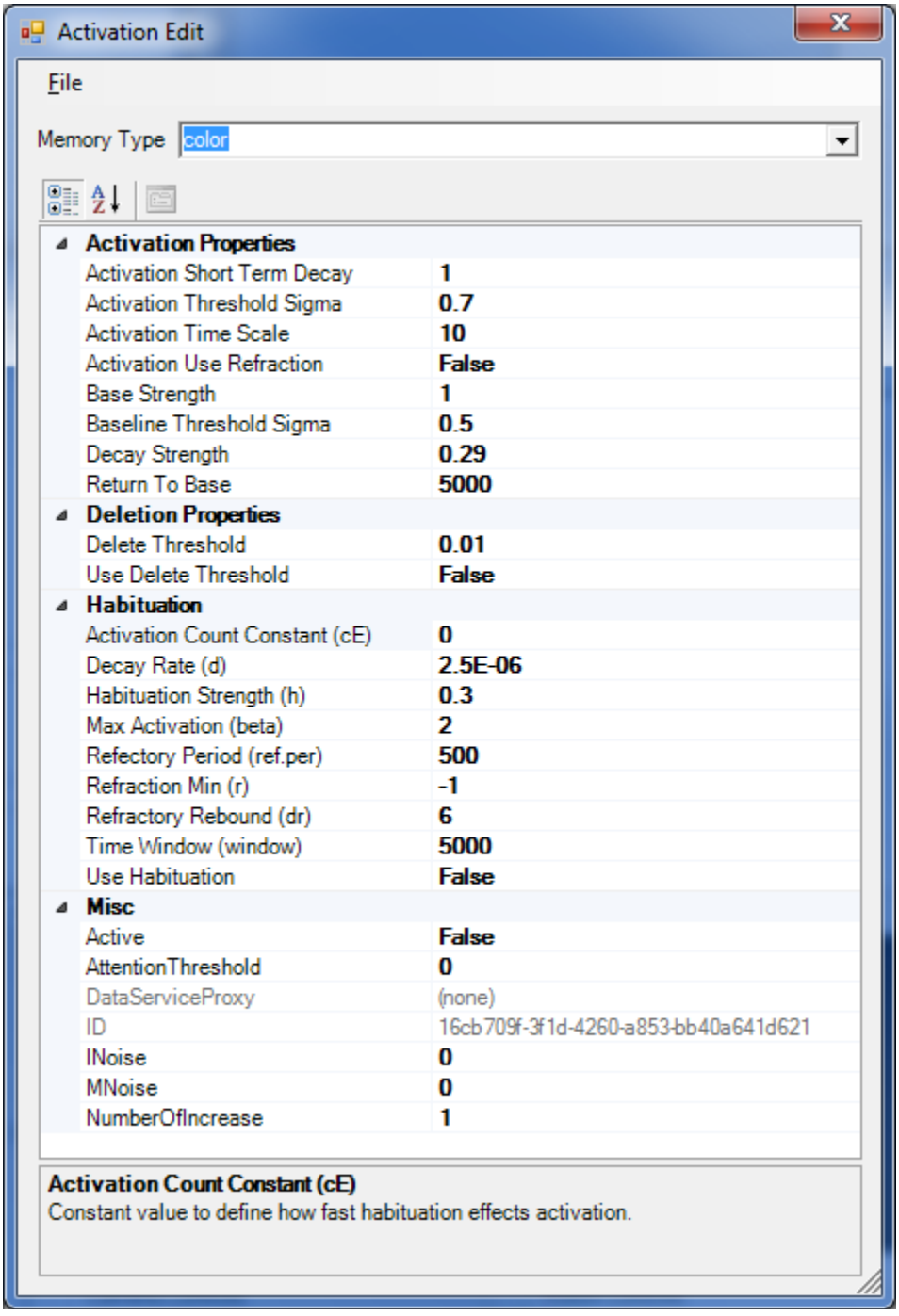


Fig. 120 Activation property page

Activation Parameters

Activation Short Term Decay
This value defines the decay behavior of refraction.

Activation Threshold Sigma
This value defines how much the activation noise can vary.

Activation Time Scale
This value affects the last time-value used in computing refraction.

Activation User Refraction
This value is a Boolean that toggles the use of refraction on and off.

Base Strength
This is the starting base strength value assigned a memory at its creation time.

Baseline Threshold Sigma
This value defines how much the baseline noise can vary.

Decay Strength
The value used to calculate how fast a memory will decay. The larger the value, the faster the memory will decay.

Return To Base
The number of activation that can occur before the memory is set back to base.

Deletion Properties

Delete Threshold
This value is used to determine when a memory can be deleted. If the strength of the memory goes below this value, it will be deleted if Use Delete Threshold is set to true.

Use Delete Threshold
When true, this value enables the deletion of a memory when it falls below the Delete Threshold.

Habituation

Activation Count Constant (cE)
This value is a constant that defines how fast habituation affects activation.

Decay Rate (d)
This value is the rate of decay.

Habituation Strength (h)
This value represents the magnitude of habituation to apply.

Max Activation (beta)
This value represents the maximum strength that can be achieved when activation occurs.

Refectory Period (ref.per)
This value represents the amount of time in milliseconds needed to refract after an activation.

Refraction Min (r)
This is the minimum value for a memory to refract to after its activation.

Refractory Rebound (dr)
This parameter adjusts the arc of activation recovery during refraction.

Time Window (window)
This parameter is the interval (window) in milliseconds used to calculate habituation strength using the number of activations in the window.

Use Habituation
This Boolean parameter determines whether or not habituation is applied to the activation calculations for the memory.

Misc

Active
This Boolean indicates whether or not the fact is an active fact in memory.

Attention Threshold
To be determined (TBD)

DataServiceProxy (not editable)
TBD

ID (not editable)
TBD

INoise
TBD

MNoise
TBD

NumberOfIncrease
TBD

9. SubSimProcessor Property Pages

Several subsims have properties that control their execution and how they handle the data they process. The following are examples of the current property change pages available. Each property listed has an explanation displayed in the text field at the bottom of the dialog that will appear when that property is highlighted. Each of the following subsims contain the base class parameters common to all subsims.

Base Class Parameters

Capture Level Specifier

Four levels of capture level are available.

- Capture: tells receiving subscriber that the target is for capture only (Template Generation subsim).
- Track: tells the receiving subscribers that the target is for tracking only (tracking subsims).
- TrackAndCapture: tells the receiving subscriber that the target is for both tracking and capturing.
- None: do nothing. Selecting this is an option during productions for suspending the disposition of the emitted targets as long as is appropriate to the purpose of the production.

Debug Level Specifier

Three levels of diagnostic reporting are available for the user who wants to regulate the flow of debug data: Silent, Normal, and Verbose.

Result Fact Type

This is the default name of the fact that gets published by various subsims containing the results of commands that can be executed on them. Each of these subsims has a command available to override it, but the default value is the name of the subsim as specified in the config file.

UseLocalVFWDevice

This property indicates whether or not the subsim will be connected to a local Video for Windows (VFW) device for image frame acquisitions. This is useful for subsims that need to garner information from a scene other than the main camera's. Notably, the facial-recognition subsims would make use of this because a specific operator might be required to be identified by the robot before it will execute commands.

Shown in Fig. 121 are properties specific to each individual subsim processor. (Subsims that do not have any user settable parameters other than base class parameters are not included.)

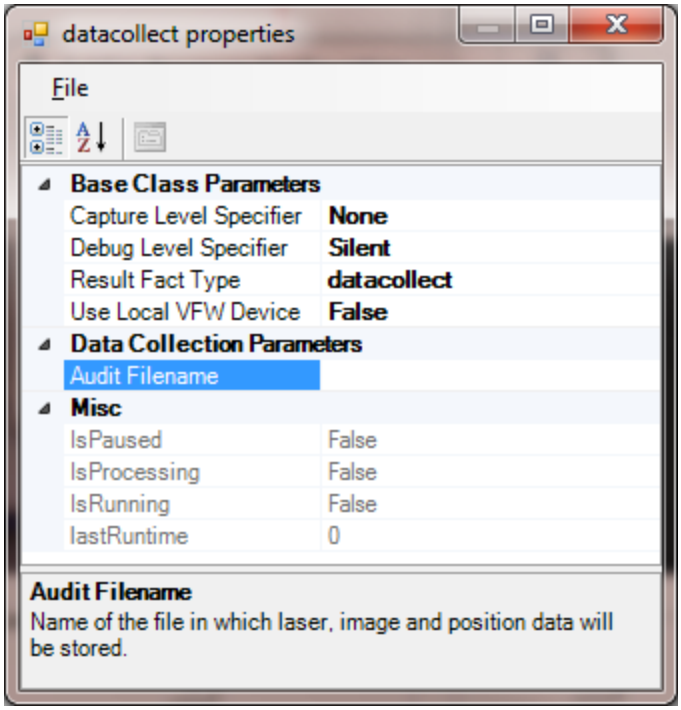


Fig. 121 Data-collection subsim properties

9.1 Data Collection

Data-Collection Parameters

Audit Filename

The name of the file into which the logged data are written.

9.2 Boredom

Boredom subsim properties are shown in Fig. 122.

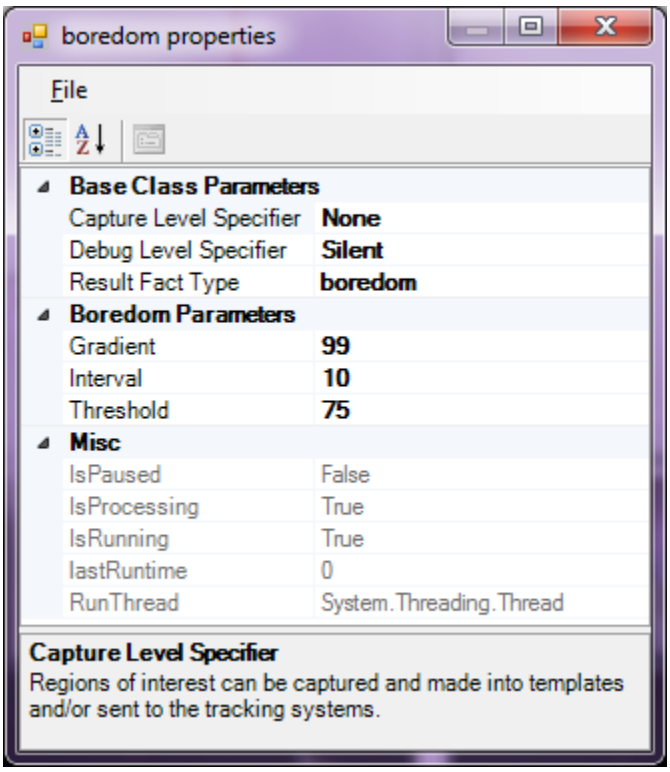


Fig. 122 Boredom subsim properties

Boredom Parameters

Boredom Interval

The number of consecutive samples over which the change in correlated laser observations is assessed.

Boredom Gradient

The value that each correlation between laser readings must reach or exceed to be considered “bored”.

Boredom Threshold

The percentage of all correlation values assessed in the time interval that have met the Boredom Gradient criterion that must be met or exceeded to classify the time interval as “boring”.

9.3 Aural Boredom

Aural Boredom subsim properties are shown in Fig. 123.

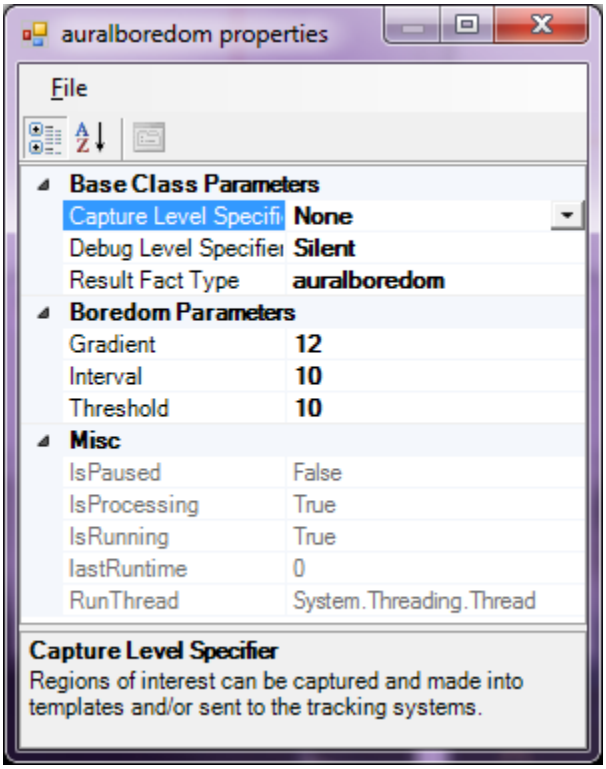


Fig. 123 Aural Boredom subsim properties

Boredom Parameters

Boredom Interval

The number of consecutive sound readings over which the change in correlated audio snippets is assessed.

Boredom Gradient

The value that each correlation between sound readings must reach or exceed to be considered “bored”.

Boredom Threshold

The percentage of all correlation values assessed in the sampling which have met the Boredom Gradient criterion that must be met or exceeded to classify the time interval as “boring”.

9.4 VisualBoredom

Visual Boredom subsim properties are shown in Fig. 124.

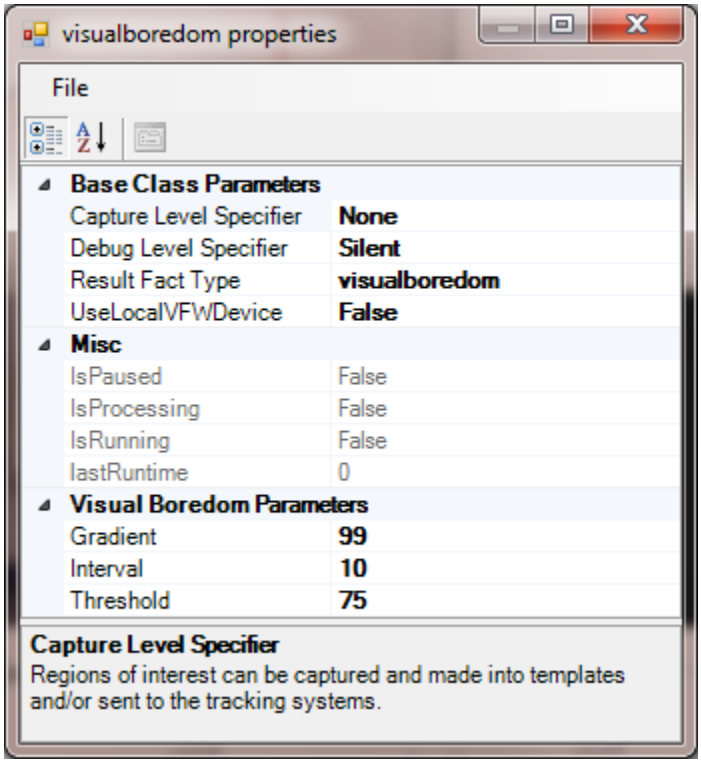


Fig. 124 Visual Boredom subsim properties

Visual Boredom Parameters

Boredom Interval

The number of consecutive images over which the change in correlated image observations is assessed.

Boredom Gradient

The value that each correlation between images must reach or exceed to be considered “bored”.

Boredom Threshold

The percentage of all correlation values assessed in the sampling that have met the Boredom Gradient criterion that must be met or exceeded to classify the time interval as “boring”.

9.5 ImageCorrelation

Image Correlation subsim properties are shown in Fig. 125.

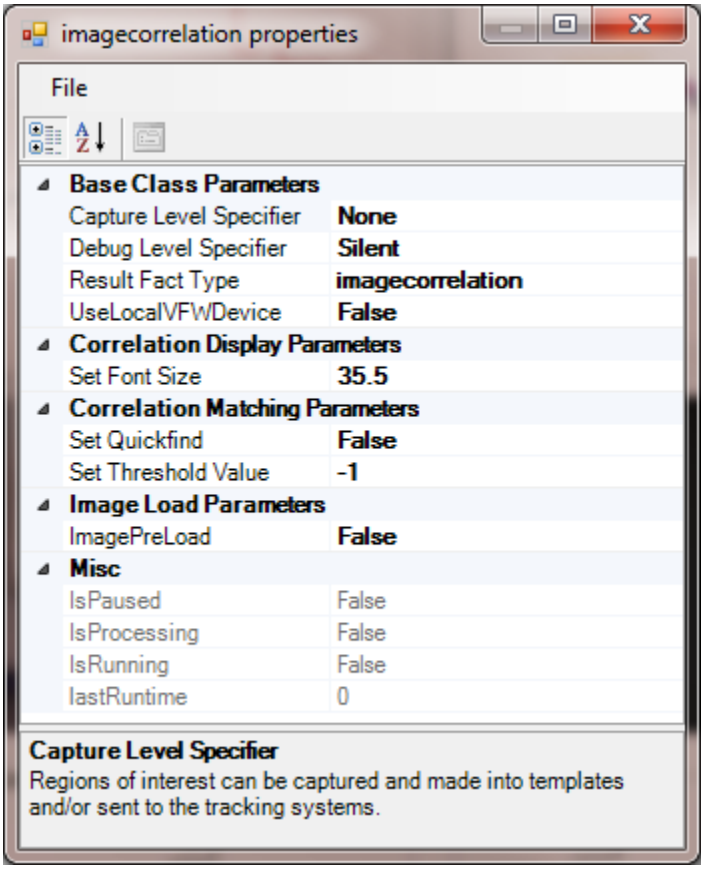


Fig. 125 Image Correlation subsim properties

Correlation Display Parameters

Set Font Size

Specifies the size of the font to be displayed. This is a useful parameter because of the different resolutions possible with the camera.

Correlation Matching Parameters

Set Threshold Value

Specifies minimum value a correlation can have before it is considered a good match.

Toggle QuickFind

When set to true, the first match that meets the Threshold Value is returned.

Image Load Parameters

ImagePreLoad

Specifies whether or not to load the images into facts for use in image matching or to leave them all on disk and just load their names and locations.

9.6 Template Matching

Template Matching subsim properties are shown in Fig. 126, and the dialog box to select bin sizes for color histogram matching is shown in Fig. 127.

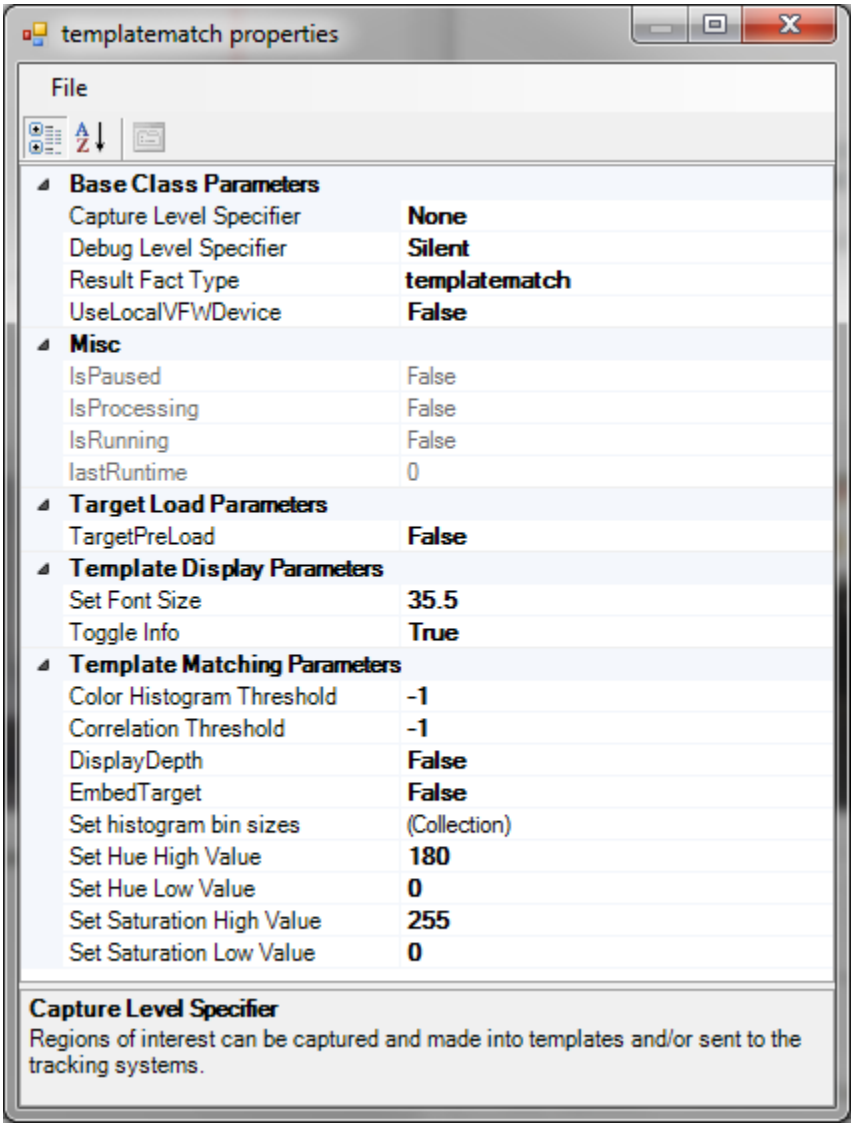


Fig. 126 Template Match subsim properties

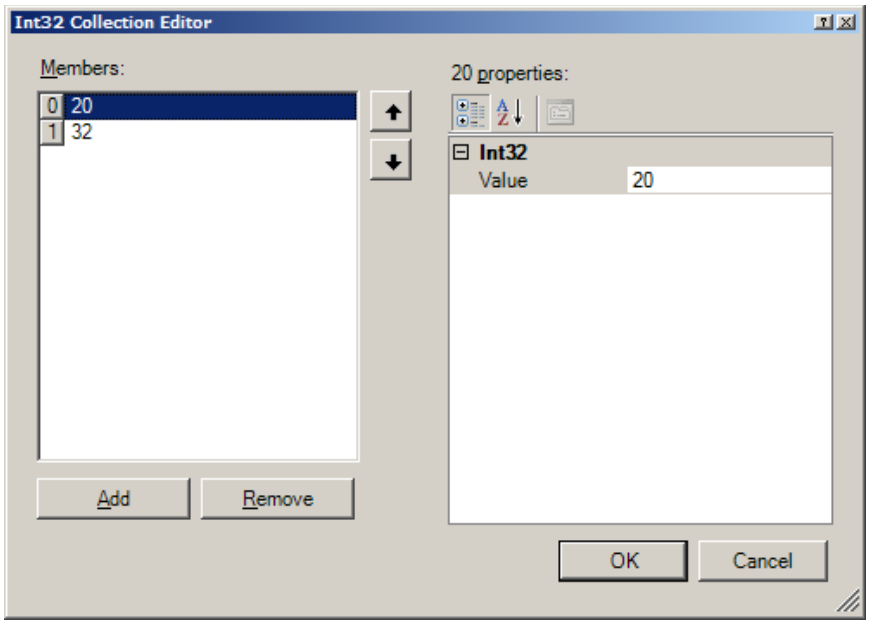


Fig. 127 Dialog to select bin sizes for color histogram matching

Template Load Parameters

TargetPreLoad

When set to true, targets will be loaded into memory when the subsim is first started. If false, the targets will be loaded in an as-needed basis. Once loaded they will remain in memory.

Template Display Parameters

Set Font Size

Allows the user to resize the font used to display information to match the chosen image size. Users should experiment with the values to suit their viewing preferences. In the future, presets will be implemented to shortcut this process.

Toggle Info

Allows the user turn display of target acquisition information on or off. The information can obscure elements of the scene, which can be undesirable at times.

Template Matching Parameters

Color Histogram Threshold

Allows the user to change the threshold at which a match is considered valid using color histogram matching. Higher numbers are theoretically more accurate while lower numbers make for a quicker match when used in conjunction with the quickfind directive described in Section 7.18.

Correlation Threshold
Allows the user to change the threshold at which a match is considered valid using correlation matching. Higher numbers are theoretically more accurate while lower numbers make for a quicker match when used in conjunction with the quickfind directive.

Display Depth
Allows the user to toggle a false color-gradient display depicting depth in the image.

Embed Target
Allows the user to toggle overlaying the target image on the area it was detected in. This allows the user to do a quick sanity check on the acquisition.

Set Histogram Bin Sizes
Allows the user to specify the sizes of the hue and saturation bins to use in color histogram matching.

Set Hue High Value
Allows the user to specify the upper value limit for the hue histogram bin.

Set Hue Low Value
Allows the user to specify the lower value limit for the hue histogram bin.

Set Saturation High Value
Allows the user to specify the upper value limit for the saturation histogram bin.

Set Saturation Low Value
Allows the user to specify the lower value limit for the saturation histogram bin.

9.7 Faces

The Faces subsim properties are shown in Fig. 128.

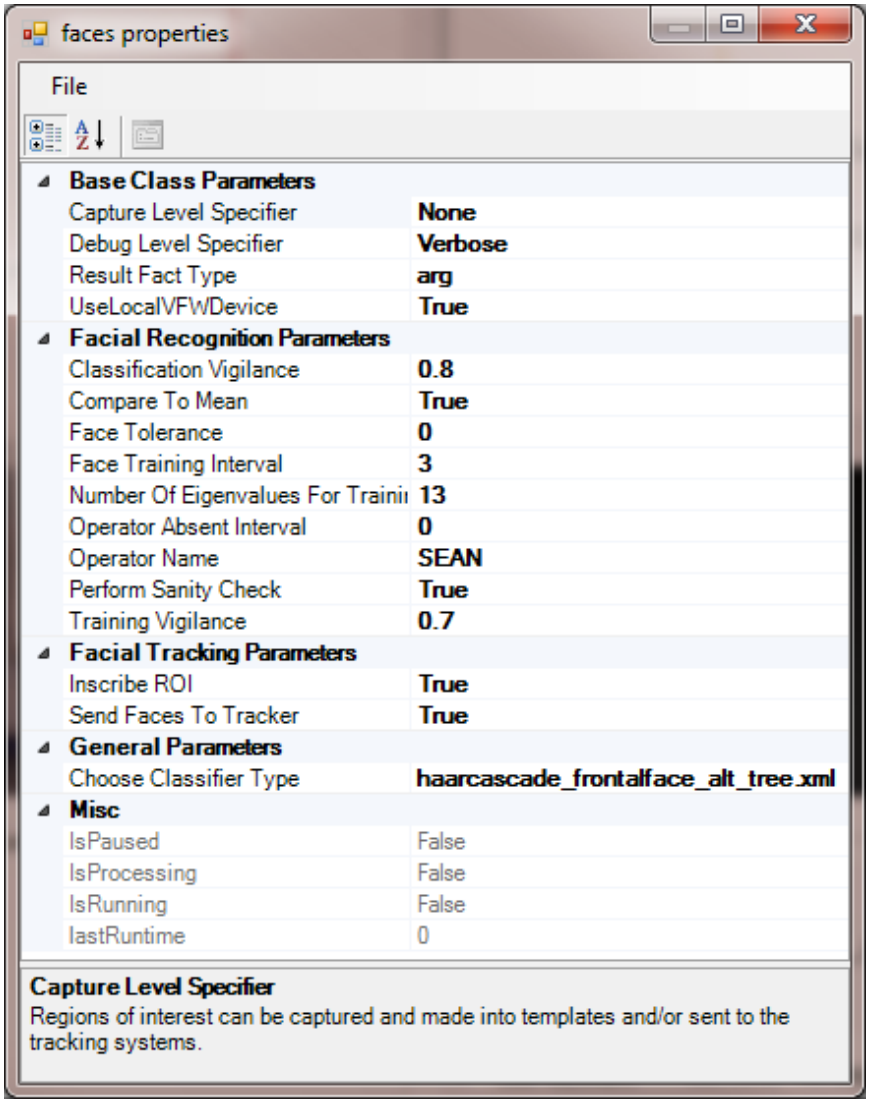


Fig. 128 Faces subsim properties

Facial Recognition Parameters

Classification Vigilance

This value governs how strict the ARTMap neural network is when performing a classification on input data. The higher the value, the stricter and, generally, the more useful.

Face Tolerance

This sets the sanity check level for false positives on the face detection. Image correlation to a known average face is used for this.

Operator Absent Interval
This value determines how long the current operator of SS-RICS can be out of the FOV before being declared unavailable.

Operator Name
This value designates the name of the current operator of SS-RICS. The operator must be learned the face detection system for this to be useful.

Perform Sanity Check
This value governs whether or not false-positive face candidates will be examined.

Training Vigilance
This value controls the number of categories created by the ARTMap neural network when it is learning. Low values tend to produce more categories, which seem to work best.

Facial Tracking Parameters

Send Faces To Tracker
This value tells the subsim to send its detected faces to the Tracking subsim. As long as this flag is on, all detected faces will be sent continuously.

General Parameters

Choose Classifier Type
This value tells the subsim which Haar classifier training set to use. The default is for faces, but full body, upper and lower body, and individual facial features are set as well.

Miscellaneous Parameters

UseLocalVFWDevice
This specifies whether or not to use the camera feed from a local VFW device instead of the one specified in the config file.

9.8 Facial Recognition

The Facial Recognition subsim properties are shown in Fig. 129.

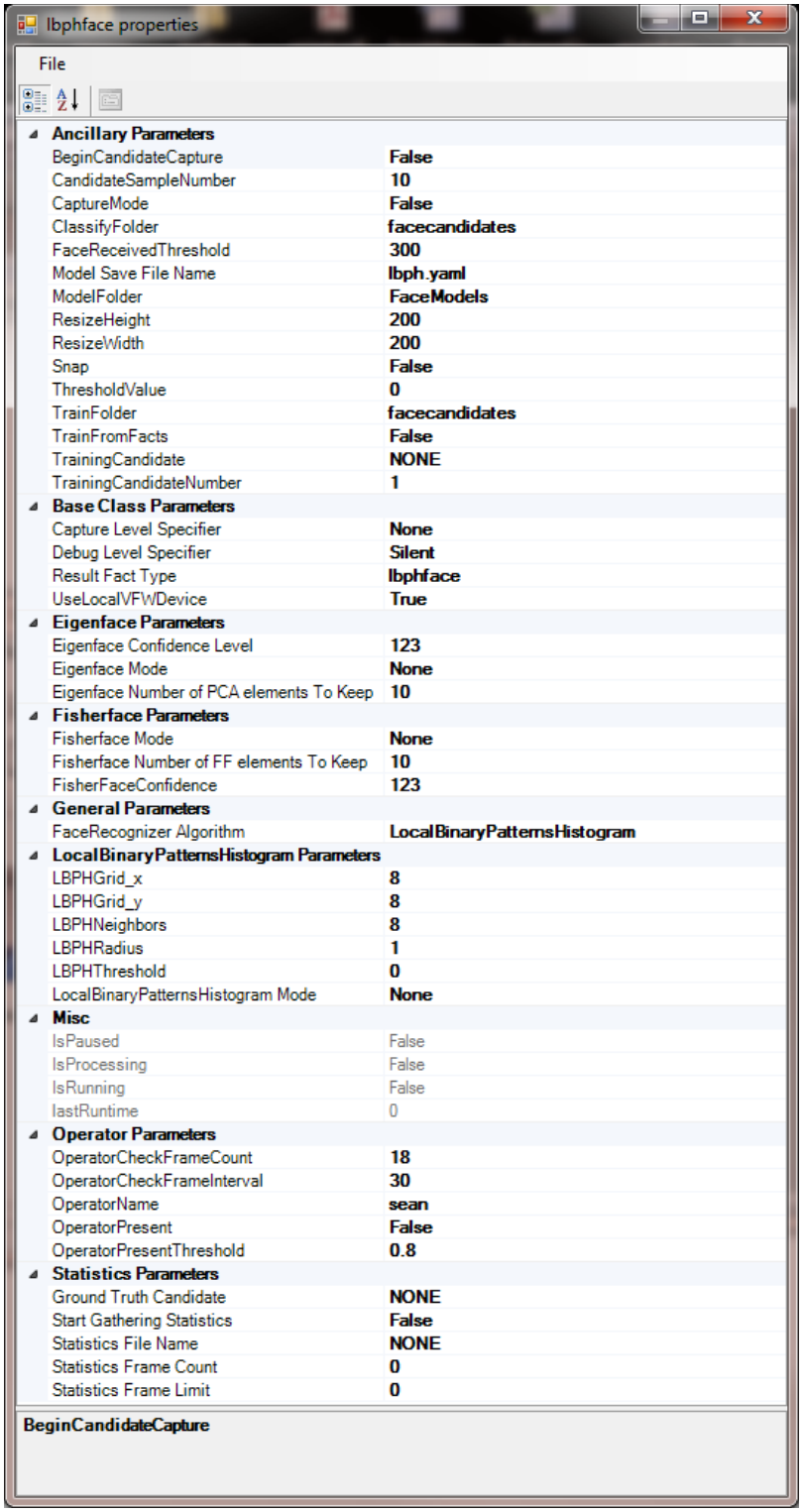


Fig. 129 Facial Recognition subsim properties

Ancillary Parameters

BeginCandidateCapture

This property, when the capture mode property is true, will start taking photographs of the candidate face every 2 s (and when detected by the Face subsim).

CandidateSampleNumber

This sets the number of snapshots to capture for the current candidate.

CaptureMode

This places the subsim in candidate capture mode.

ClassifyFolder

This is the location from which the subsim will get photos to use in conducting a sanity check on the model. This entails training it and then using each face used in training to test its classification ability. Each model tested 100% accurate against each face. (Not used in normal operations.)

FaceRecievedThreshold

This is the amount of time that needs to pass before the subsim will assume the last face sent to it from the Faces subsim is stale. The Faces subsim will lose track of a face if it turns too far askance or exits the FOV altogether. As each face is received, a timer is started to determine the current face's age. If it exceeds the threshold, the current face will be rendered invalid.

Model Save File Name

This is the file name used to store the trained model if the operator so wishes to save it. Once saved, it can be reloaded, which is the fastest way to effect training.

ModelFolder

This is the file name of the folder (subfolder of working directory unless fully qualified) used to store the subsim's model save/load files. It is set to NONE initially, meaning the models will be stored in the program's current working directory.

ResizeHeight

This is the number of pixels high (rows) the candidate training and categorizing images will be resized to.

ResizeWidth

This is the number of pixels wide (columns) the candidate training and categorizing images will be resized to. The 2 values, height and width, should be the same, resulting in a square since candidate capture is always a square image. Values of 200 respectively appears optimal given a camera image size of 640 × 480 pixels.

Snap
When Capture mode is on, this will, when set to true, take a photo of the the current subject (provided a valid face is present). Once set to true, it resets to false once the picture is taken.

ThresholdValue
This value, when set, will set the bar for recognition by the algorithm. If the prediction value exceeds this value, the candidate will be deemed unrecognizable and return a value of –1.

TrainFolder
This is the location the subsim will place the pictures taken of candidates for future loading.

TrainingCandidate
This is the name of the current candidate whose sample photos will be or are being taken for training the model. It is stored in a facecandidate fact, and when the model is saved, it is stored in a file parallel to the model save file, which is the same name as the model file but with the added extension of .names. So if the model file is lbph.yaml, the corresponding name file will be lbph.yaml.names. The reason this file exists is because the software package from openCV, which is the core of the subsim, uses integers to identify individual candidates for training and classification.

TrainingCandidateNumber
This is the number of the current candidate whose sample photos will be or are being taken for training the model. It is stored in the model once it is trained and is saved in the model file. It is the native identifier for each subject trained and classified and must be unique for each. It cannot be 0 or negative and thus can begin at 1.

UseLocalVFWDevice
Because the facial recognition subsim is used primarily to identify an individual as being its operator, the main camera would be ill-served if trained only on the person rather than free to assess the environment. An additional VFW device can be attached and used in addition to the main camera for this expressed purpose.

Eigenface Parameters

Eigenface Confidence Level
This value sets the maximum distance the nearest neighbor can be to be considered valid when attempting to classify a candidate face. For Eigenface, this is typically around 16,000. The default is DBL_MAX.

Eigenface Mode
This value sets the mode for initializing the Eigenface algorithm. Its values are the following:

- None
- SpecifyPCA2Keep
- SpecifyConfidence
- SpecifyBoth

None means use all PCA components generated and set tolerance to maximum. SpecifyPCA2Keep means to keep the number of PCA components as specified in the property NumberOfPCA2Keep. SpecifyConfidence means use the value stored in the property to preset the model's confidence level (it and all other model's confidence levels can be changed dynamically after initialization). SpecifyBoth means use both above mentioned values to intialise the model.

Eigenface Number Of PCA Elements To Keep
This value sets the number of PCA components you want the model to use in training and classifying a candidate face. Typically, around 10 or more are considered desirable. The default is all components.

Fisherface Parameters

Fisherface Confidence Level
This value sets the maximum distance the nearest neighbor can be to be considered valid when attempting to classify a candidate face. For Fisherface, this is typically around 16,000. The default is DBL_MAX.

Fisherface Mode
This value sets the mode for initializing the Fisherface algorithm, and has the following values:

- None
- SpecifyFF2Keep
- SpecifyConfidence
- SpecifyBoth

None means use all PCA components generated and set tolerance to maximum. SpecifyFF`2Keep means to keep the number of PCA components as specified in the property NumberOfFF2Keep. SpecifyConfidence means use the value stored in the property to preset the model's confidence level (it and all other model's

confidence levels can be changed dynamically after initialization). SpecifyBoth means use both above mentioned values to intialise the model.

Fisherface Number of FF Elements to Keep
This value sets the number of Fisher Face components you want the model to use in training and classifying a candidate face. Typically, around 10 or more are considered desirable. The default is all components.

General Parameters

FaceRecognizer Algorithm
This value reflects the algorithm employed by this instance of the subsim.

LocalBinaryPatternsHistogram Parameters

LBPHGrid_x
This specifies the number of patches in the x direction, set to 8 by default.

LBPHGrid_y
This specifies the number of patches in the y direction, set to 8 by default.

LBPHNeighbors
This specifies the number of pixels surrounding each pixel being compared with that pixel, and is considered the neighborhood of that pixel. This is set to 8 by default.

LBPHRadius
This specifies how far away from the pixel being examined to look for neighbors, set to 1 by default.

LBPHThreshold
This specifies the value to compare against when thresholding each pixel against its neighbors, set to 256 by default.

With LBPH, the default values are best. Increasing any of them will result in a noticeable degradation in performance as they represent exponentially affected components.

Operator Parameters

OperatorCheckFrameCount
The current frame count while completing a cycle for operator-presence verification.

OperatorCheckFrameInterval
This specifies the number of frames to scan in a row for the presence of the operator.

OperatorName
This specifies the name of the person designated as the operator.

OperatorPresent
This specifies whether or not, during the last frame interval, the operator was found a specified percentage of frames.

OperatorPresentThreshold
This specifies the percentage of frames out of the frame interval specified wherein the operator was correctly identified. If the operator was found at or above that percentage, the presence of the operator is acknowledged on the subsim's processed image.

Statistics Parameters

Ground Truth Candidate
The actual identity of the candidate being classified.

Start Gathering Statistics
A value of true tells the subsim to begin gathering statistics.

Statistics File Name
This specifies the name of the file to store the statistics in. It is a comma separated file suitable for opening in Excel.

Statistics Frame Count
This specifies the current number of frames examined for the statistics.

Statistics Frame Limit
This specifies the number of consecutive frames to attempt classification of a candidate.

9.9 LineSegment

Figure 130 shows the LineSegment subsim properties.

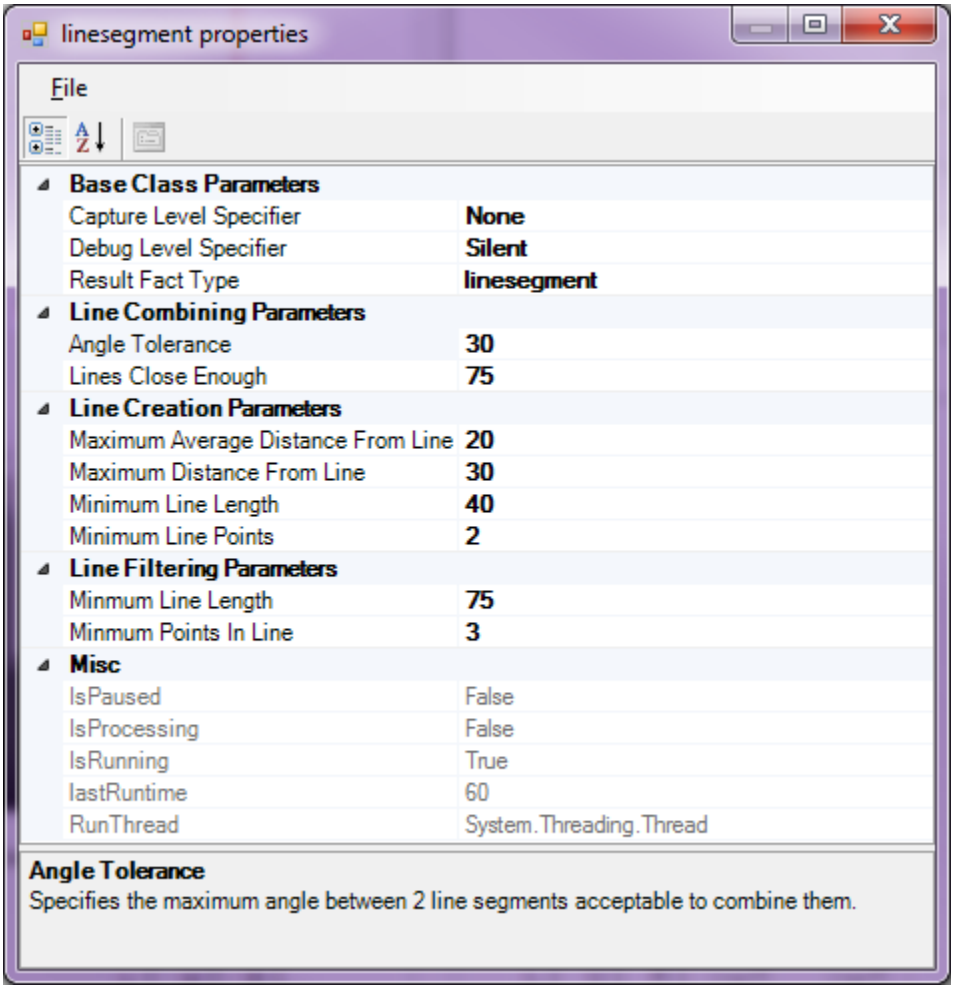


Fig. 130 LineSegment subsim properties

Line Creation Parameters

Maximum Average Distance From Line

The maximum over time distance a point can be to a line to consider it part of the line.

Maximum Distance from Line

The maximum distance a point can be to a line to consider it part of the line.

Minimum Line Length

The minimum line length required before a set of points are joined into a line segment.

Minimum Line Points
The minimum number of points required before the points are joined into a line segment.

Line Combining Parameters

Angle Tolerance
The maximum angle between 2 line segments that are to be merged into one line segment.

Line Close Enough
The distance between 2 line segments that are to be merged into one line segment.

Line Filtering Parameters

Minimum Points in Line
The minimum points in a line that may be accepted as a valid line segment.

Minimum Line Length
The minimum length of a line that may be accepted as a valid line segment.

9.10 Line

Figure 131 shows the Line subsim properties.

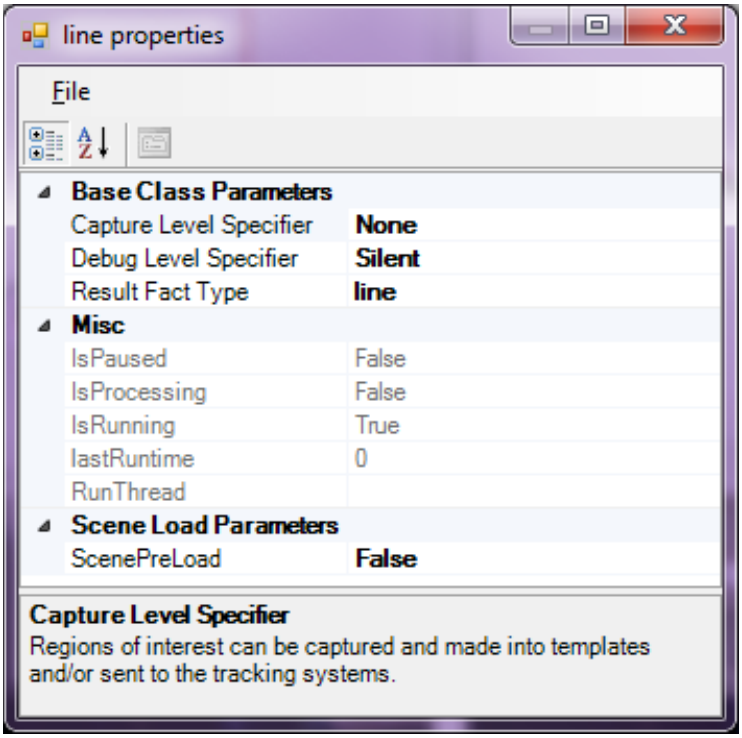


Fig. 131 Line subsim properties

Scene Load Parameters

ScenePreLoad

This property controls whether or not the subsim will load scenes with or without laser data upon subsim startup. Preloading takes more memory than not preloading. As scenes are accessed to fulfill comparisons requests, these values will be loaded and stay loaded.

9.11 Gap

Figure 132 shows the Gap subsim properties.

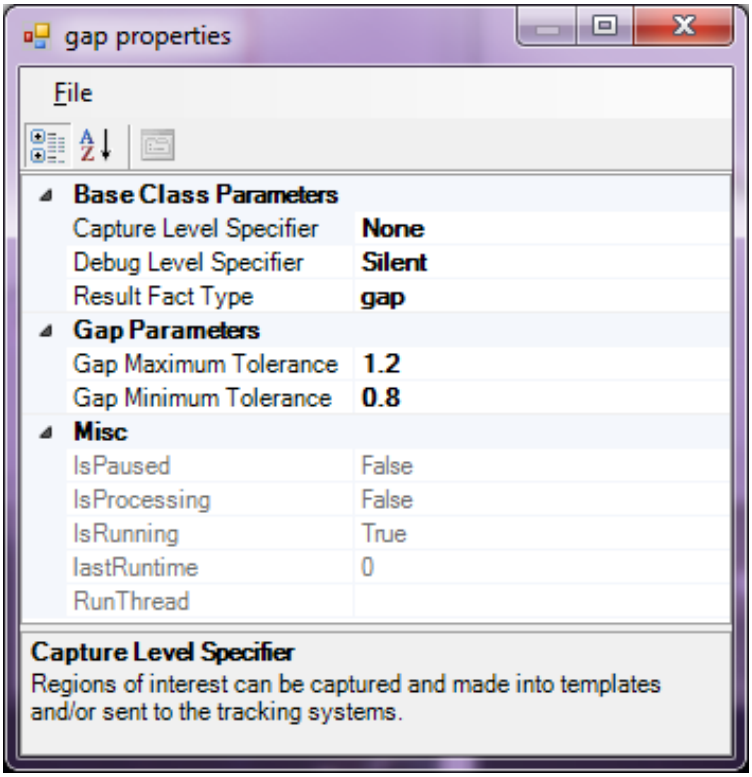


Fig. 132 Gap subsim properties

Gap Parameters

Gap Maximum Tolerance

This property controls the maximum distance between 2 line segments that can be designated a gap.

Gap Minimum Tolerance

This property controls the minimum distance between 2 line segments that can be designated a gap.

9.12 SceneDepth

Figure 133 shows the Gap subsim properties.

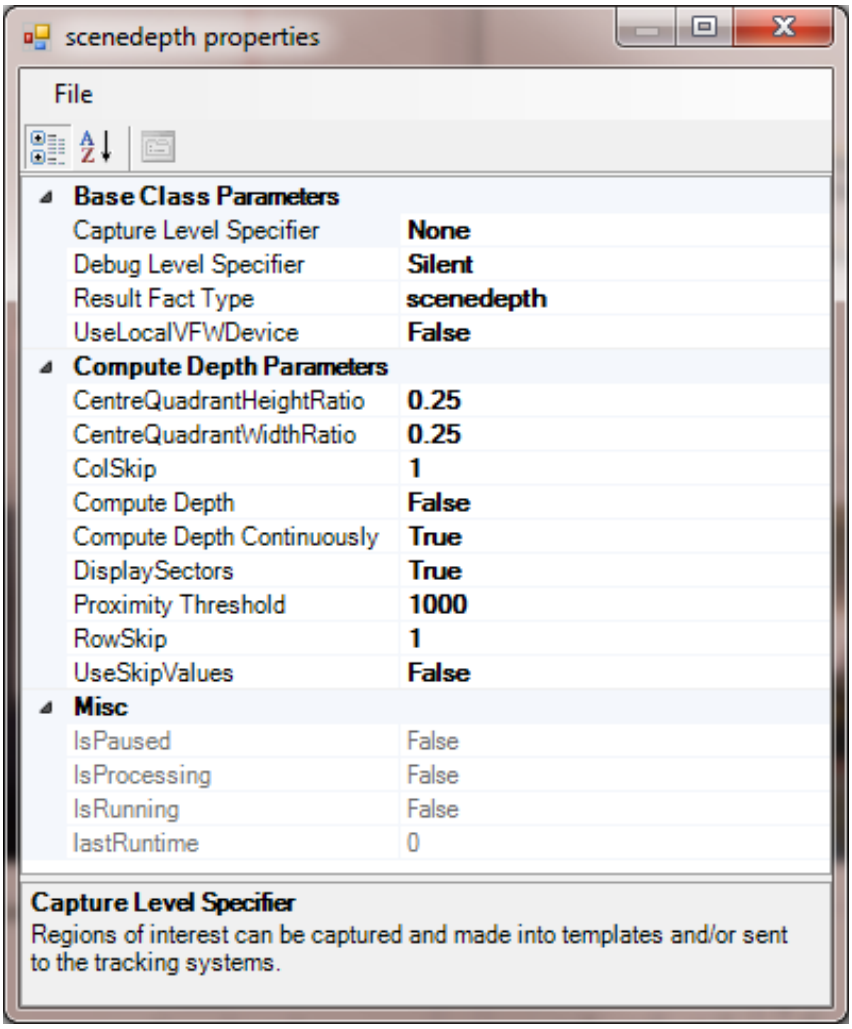


Fig. 133 Gap subsim properties

Compute Depth Parameters

CentreQuadrantHeightRatio

This property controls the ratio of the screen height to which the height of the center sector will be set.

CentreQuadrantWidthRatio

This property controls the ratio of the screen width to which the width of the center sector will be set.

Compute Depth (not used normally)

This property does a one-off computation of the depths of the scene and publishes them in a fact.

Compute Depth Continuously
This property computes the average depths in the scene from frame to frame and updates the result fact at the same rate.

Display Sectors
This property, when set to true, draws the sectors on the screen.

Proximity Threshold
This property governs the distance at which an average depth can be considered far or near.

ColSkip
This property governs the number of columns to skip between assessing the depth of each pixel. When using the virtual environment, the cost of depth calculation per pixel is high, so skipping every few pixels gives a good approximation while speeding up the process.

RowSkip
This property governs the number of rows to skip between assessing the depth of each pixel. When using the virtual environment, the cost of depth calculation per pixel is high, so skipping every few pixels gives a good approximation while speeding up the process.

UseSkipValues
This property governs whether or not row/column skipping is engaged.

9.13 Moments

Figure 134 shows the Moments subsim properties.

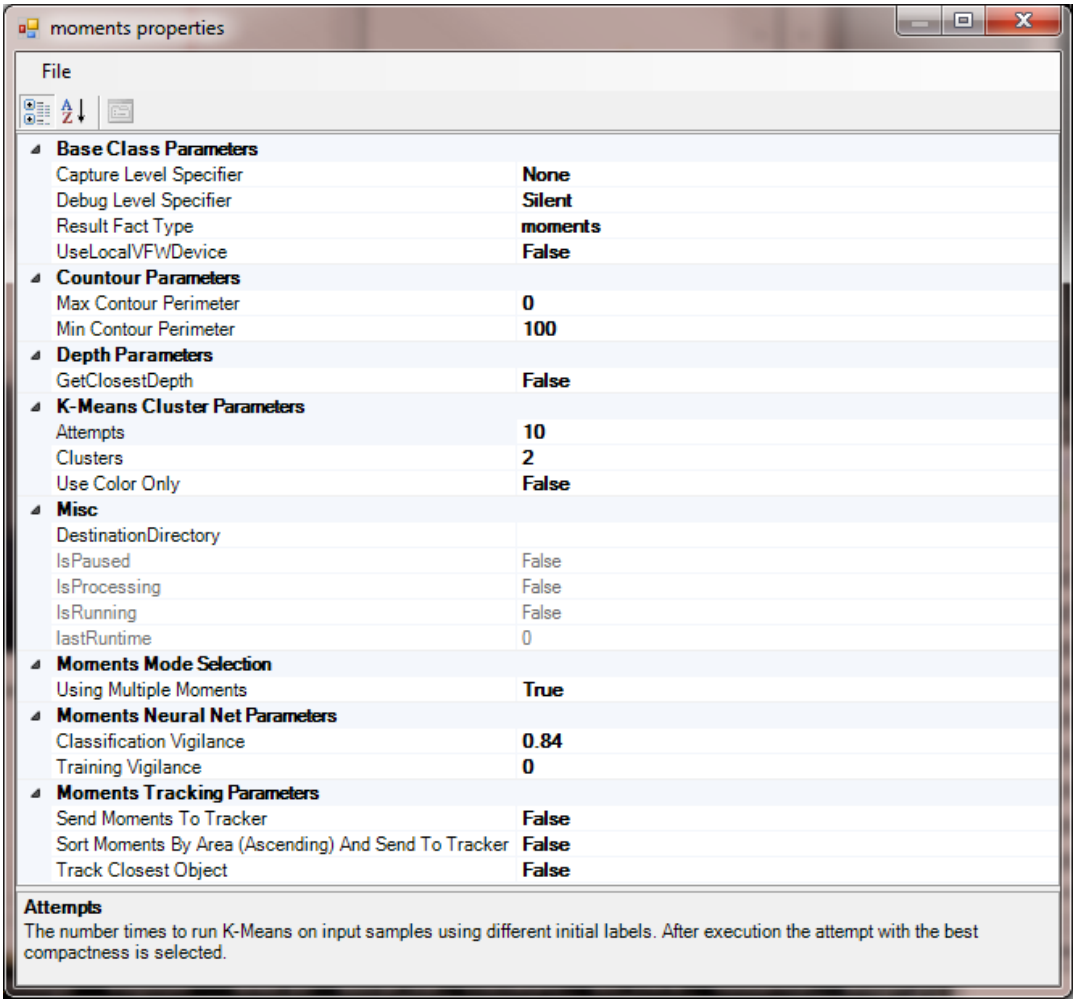


Fig. 134 Moments subsim properties

Moments Mode Selection

Using Multiple Moments

This value lets you set which mode you want: a single set or multiple sets of moments/ROIs.

Moments Neural Network Parameters

Classification Vigilance

This value governs how strict the ARTMap neural network is when performing a classification on input data. The higher the value, the stricter and, generally, the more useful.

Training Vigilance
This value controls the number of categories created by the ARTMap neural networkk when it is learning. Low values tend to produce more categories, which seem to work best.

Miscellaneous

DestinationDirectory
This value sets the name of the subdirectory (under Targets/) that will contain the regions sent to the templatematch subsim for capture.

Moments Tracking Parameters

Send Moments to Tracking
This value toggles the sending of Moments bounding boxes to the Tracking subsim. As long as it is set to true, Moments will be sent to Tracking. (Tracking will discard any moments sent that exceed the number of trackers permitted by that subsim.)

Sort Moments by Area
This value toggles the sorting of Moments bounding boxes by area so that the smallest can be sent first to the Tracking subsim.

9.14 Motion Detection

Figure 135 shows the Motion Detection subsim properties.

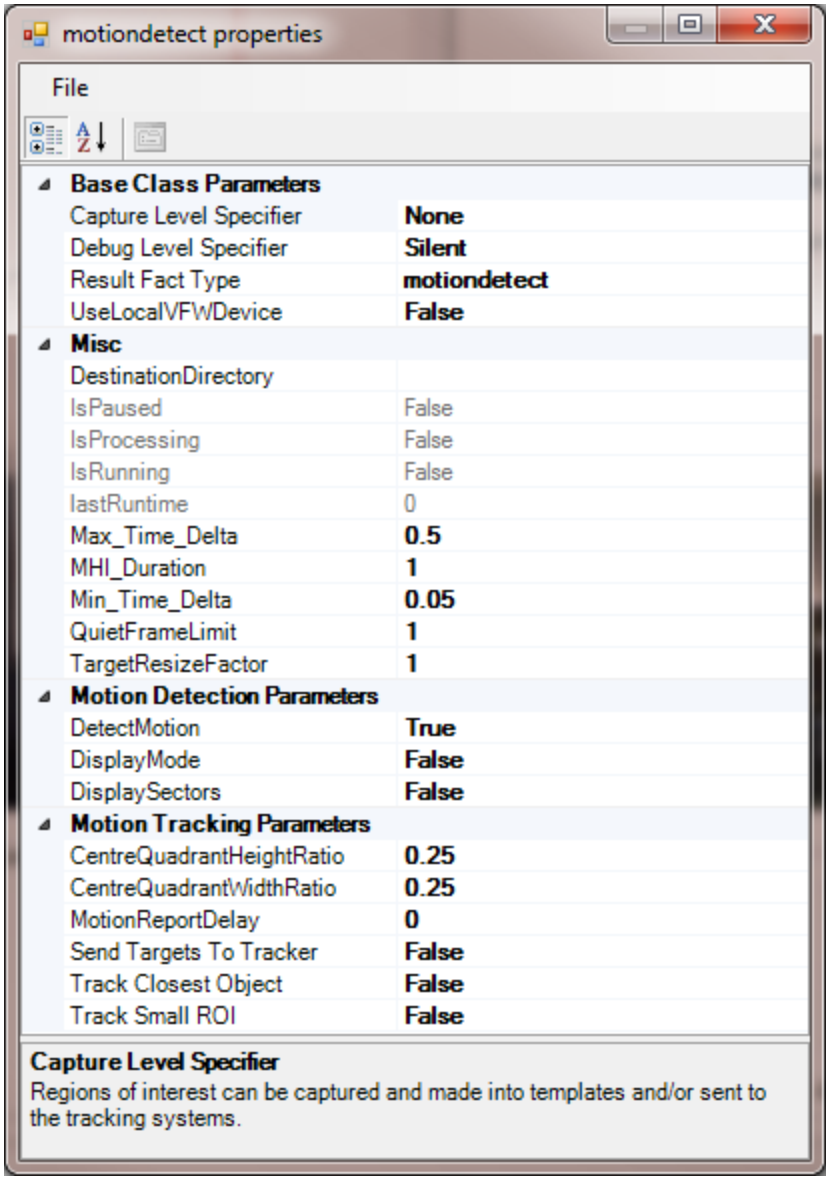


Fig. 135 Motion Detection subsim properties

Motion Detection Parameters

DetectMotion

This value allows the user to pause motion detection. This can be useful when tracking something and the robot is moving.

DisplayMode

This value controls the visual output to the Processed Image View. True means to display only the blue-shaded differences between consecutive frames, and false means display the normal image.

Misc

DestinationDirectory

This value sets the name of the subdirectory (under Targets/) that will contain the regions sent to the templatematch subsim for capture.

Motion Tracking Parameters

Send Targets ToTracker

This value tells the subsim to send detected motion bounding boxes to the Tracking subsim.

Track Closest Object

This value tells the subsim to send only the single closest of the currently detected bounding boxes to the Tracking subsim.

Track Small ROI

This value tells the subsim to send detected-motion bounding boxes reduced in size to 24 × 24 pixels. This reduces the surface area being tracked to exclude extraneous clutter from tracking.

9.15 Saliency

Figure 136 shows the Saliency subsim properties.

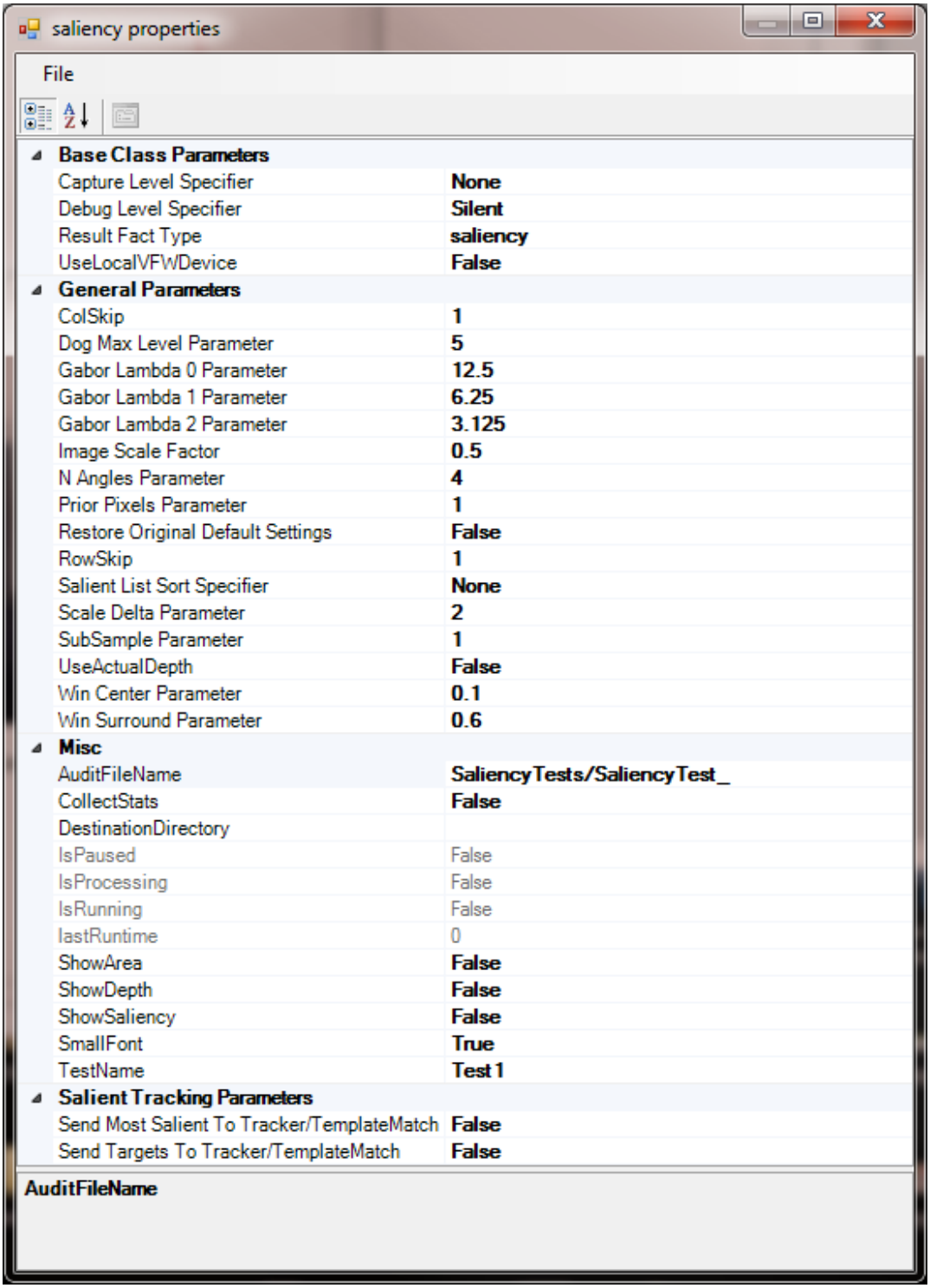


Fig. 136 Saliency subsim properties

Miscellaneous

DestinationDirectory

This value sets the name of the subdirectory (under Targets/) that will contain the regions sent to the Template Matching subsim for capture.

Salient Tracking Parameters

Send Targets to Tracker/Templatematch

When set to true, the Saliency subsim will start emitting targets to the subscribers listed in the config file. Typically and currently these are the tracking subsims and the template matcher.

General Parameters

Image Scale Factor

This value scales down the input image by the specified amount on width and height to the default: 0.5 reduces a 640- × 480-pixel image to 320 × 240. This increases processing speed but can result in some loss of precision. The value is capped at 1.0.

Salient List Sort Specifier

Controls the order that salients are presented in the comma-separated list that is published in the salients slot of the currentview/state fact. Options include the following:

- Area: sorts the list by square pixel area
- Depth: If depth data are available, the distance to the target will be used to sort by. (If the distance is specified but no depth is available, it is ignored and no sorting takes place.)
- Saliency: sorts the list by the returned saliency of each region
- None: no sorting

Dog Maximum Level parameter

Sets this saliency object constructor parameter to the user-specified value. The default is 5.

Gabor Lambda 0 Parameter

Sets this saliency object constructor parameter to the user-specified value. The default is 12.5.

Gabor Lambda 1 Parameter

Sets this saliency object constructor parameter to the user-specified value. The default is 6.25.

Gabor Lambda 2 Parameter
Sets this saliency object constructor parameter to the user-specified value. The default is 3.125.

Nangles Parameter
Sets this saliency object constructor parameter to the user-specified value. The default is 4.

Prior Pixels Parameter
Sets this saliency object constructor parameter to the user-specified value. The default is 1.

Scale Delta Parameter
Sets this saliency object constructor parameter to the user-specified value. The default is 2.

SubSample Parameter
Sets this saliency object constructor parameter to the user-specified value. The default is 1.

Win Center Parameter
Sets this saliency object constructor parameter to the user-specified value. The default is 0.1.

Win Surround Parameter
Sets this saliency object constructor parameter to the user-specified value. The default is 0.6.

Restore Original Default Settings
Resets all of the constructor parameters to the original settings the code came with.

9.16 Segmentation

Figure 137 shows the Segmentation subsim properties.

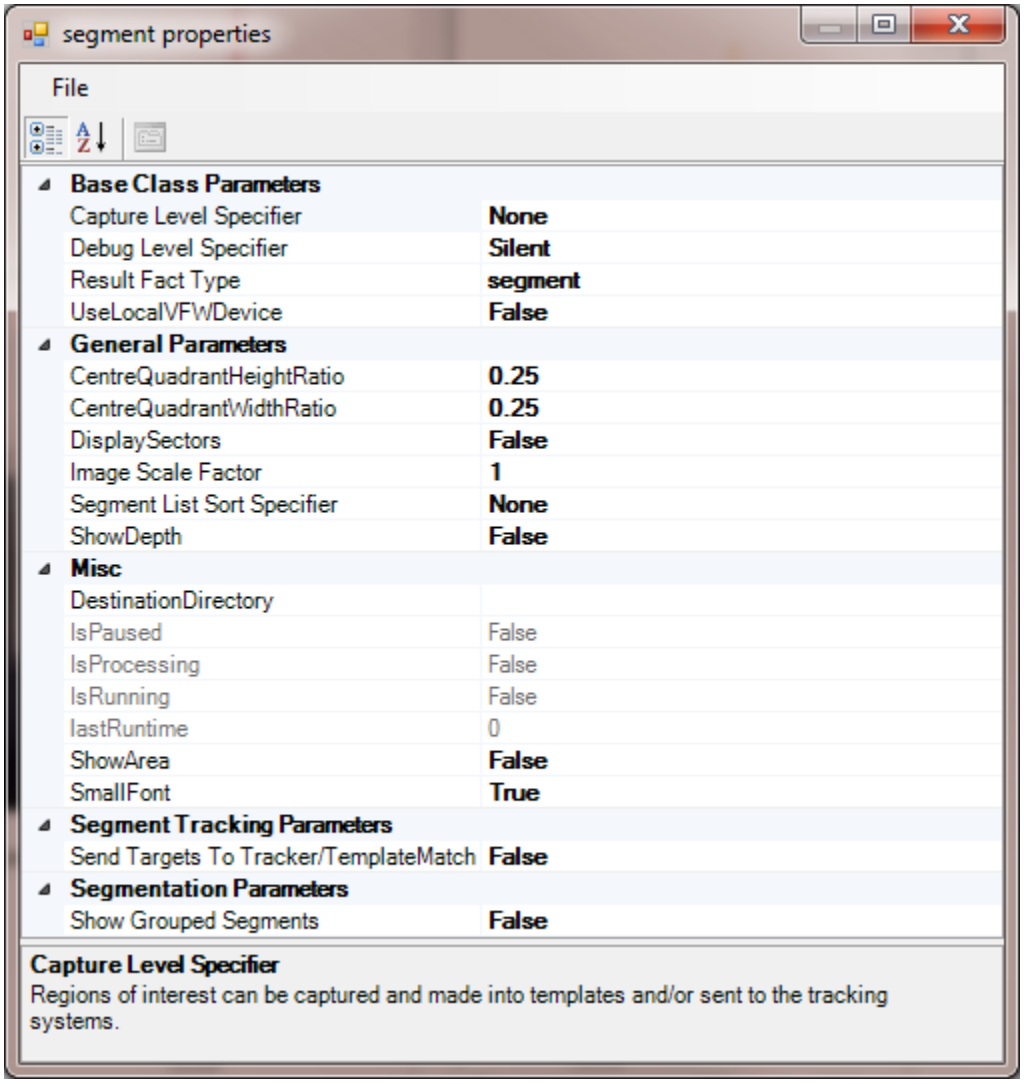


Fig. 137 Segmentation subsim properties

Miscellaneous

DestinationDirectory

This value sets the name of the subdirectory (under Targets/) that will contain the regions sent to the Template Matching subsim for capture.

ShowArea

This value displays each segments area in square pixels.

SmallFont

This value tells the subsim to use small fonts for display values in the image.

Segmentation Parameters

Show Grouped Segments

This property will combine individual segments that have been classified as the same. They will use a single color rather than individual colors.

Segment Tracking Parameters

Send Targets to Tracker/TemplateMatch

This property, depending on the capture type specified, will send segments to the tracking and/or template matching subsims.

General Parameters

CentreQuadrantHeightRatio

This property controls the ratio of the screen height to which the height of the center sector will be set.

CentreQuadrantWidthRatio

This property controls the ratio of the screen width to which the width of the center sector will be set.

Image Scale Factor

This value scales down the input image by the specified amount on width and height to the default: 0.5 reduces a 640- × 480-pixel image to 320 × 240. This increases processing speed but can result in some loss of precision. The value is capped at 1.0.

ShowDepth

This value displays each segments depth value in millimeters.

Segment List Sort Specifier

Controls the order that salients are presented in the comma-separated list that is published in the segments slot of the currentview/state fact. Options include the following:

- Area: sorts the list by square pixel area.
- Depth: If depth data are available, the distance to the target will be used to sort by. (If the distance is specified but no depth is available, it is ignored and no sorting takes place.)
- None: no sorting.

9.17 Points

Figure 138 shows the Points subsim properties.

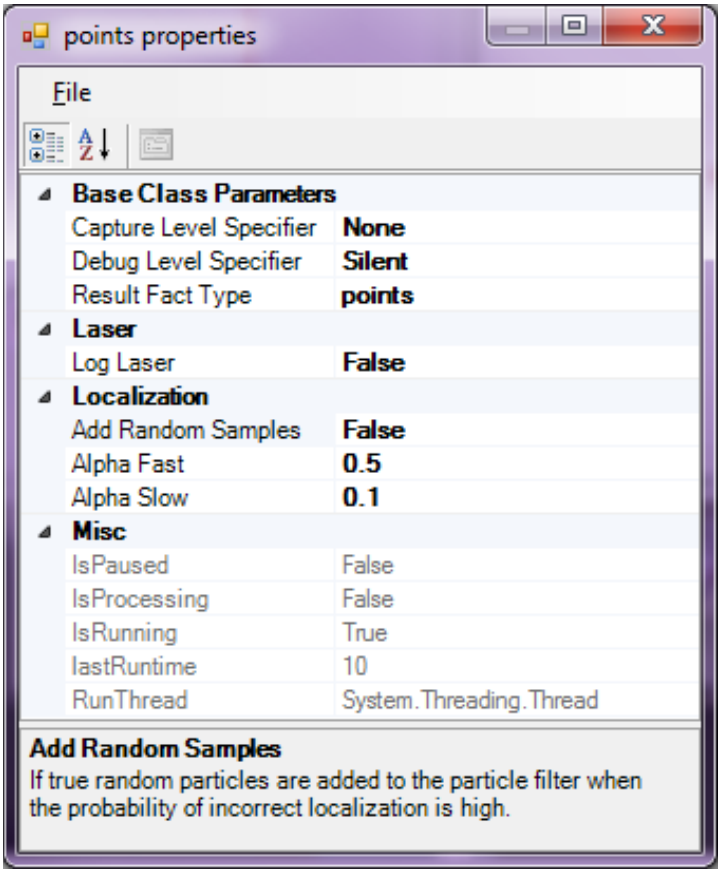


Fig. 138 Points subsim properties

Laser

Log Laser

This value toggles the logging of laser data to a file named SickLog.log.

Add Random Samples

If true, random particles are added to the particle filter when the probability of incorrect localization is high.

Localization

Alpha Fast

Adjusts the fast running average used to add random particles.

Alpha Slow

Adjusts the slow running average used to add random particles.

9.18 PredatorTLD

Figure 139 shows the PredatorTLD subsim properties.

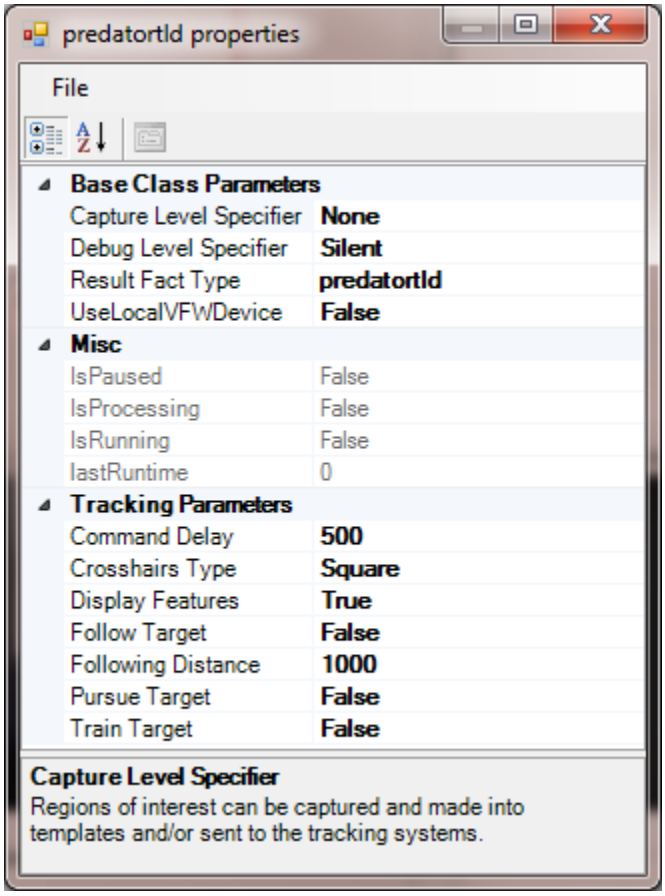


Fig. 139 PredatorTLD subsim properties

Tracking Parameters

commandDelay

This value adjusts the wait time between commands to move and adjust the camera postion when the robot is pursuing a target. This value is used in the experimental code added to show this capability is viable in the robot. It will be superseded by new parameters once this capability is implemented in a more mature form.

CrosshairsType

This value allows you to pick between 3 methods of outlining the target: rectangular bounding box, traditional crosshairs, and convexhull perimeter.

followingDistance

Adjusts the minimum distance the robot will follow when this mode is engaged. This parameter is used as part of the ability described under commandDelay and will be supplanted when the feature is eventually implemented.

followTarget
Also associated with the capability described under commandDelay, this parameter tells the robot to change its camera and body orientation to keep the tracked object in the center of its FOV.

pursueTarget
Also associated with the capability described under commandDelay, this parameter tells the robot to follow the target it is tracking using the distance specified in followingDistance.

showPoints
This value toggles the display of the tracking points, which can be useful, as they are a bit of a distraction sometimes.

9.19 Tracking

Figure 140 shows the Tracking subsim properties.

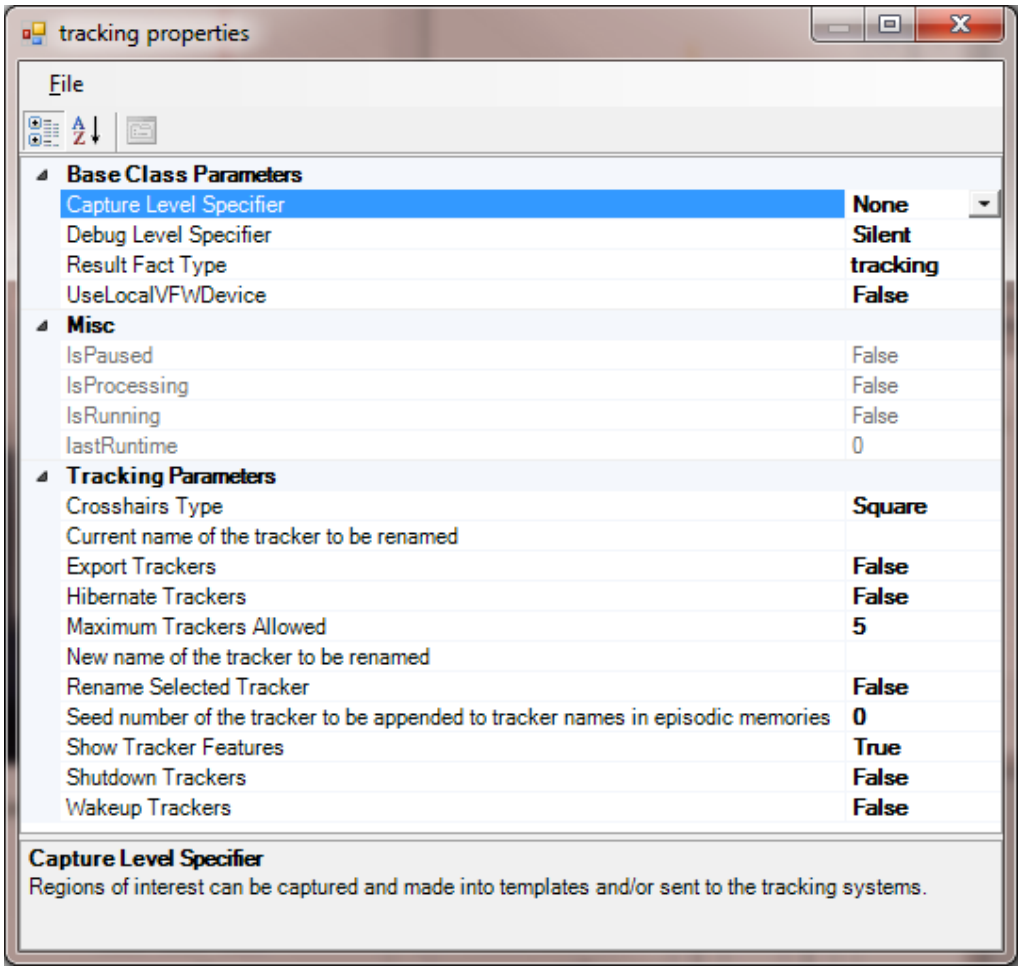


Fig. 140 Tracking subsim properties

Tracking Parameters

Crosshairs Type

This value allows you to pick between 3 methods of outlining the target: rectangular bounding box, traditional crosshairs, and convex hull perimeter.

Export Trackers

This flag tells the Tracking subsim to export (serialize and save to disk) all targets currently being tracked. It is reset to false once this is done.

Hibernate Trackers

This flag tells the Tracking subsim to suspend all tracking.

Maximum Trackers Allowed

This parameter sets the limit for how many targets can be tracked simultaneously. This limit is dependent upon how much memory is available and how many cores the processor has. The memory footprint of tracking objects (.tld files on disk) is typically approximately 5–9 MB.

New Name of the Tracker to be Renamed

This parameter sets the new name for a tracking object that will be renamed.

Current Name of the Tracker to be Renamed

This parameter sets the current name for a tracking object that is to be renamed.

Seed Number of the Tracker to be Appended to Tracker Names in Episodic Memories

This parameter is appended to the name given to a target that gets sent to the Tracking subsim from the Episodic Memory subsim. The actual name given is a datetime stamp contained in the slot Date of the fact that transported the target definition to the tracker.

Rename Selected Tracker

This flag tells the Tracking subsim to rename the tracker whose current name is oldName to the value contained in newName.

Show Tracker Features

This flag tells the Tracking subsim to display the tracking points in the bounding box or other crosshair selection. They can be distracting, so False will hide them.

Shutdown Trackers

This flag tells the Tracking subsim to shut down all active trackers.

Wakeup Trackers

This flag tells the Tracking subsim to reactivate all hibernating trackers.

9.20 Color Tracking/Detection

Figure 141 shows the Color Tracking subsim properties.

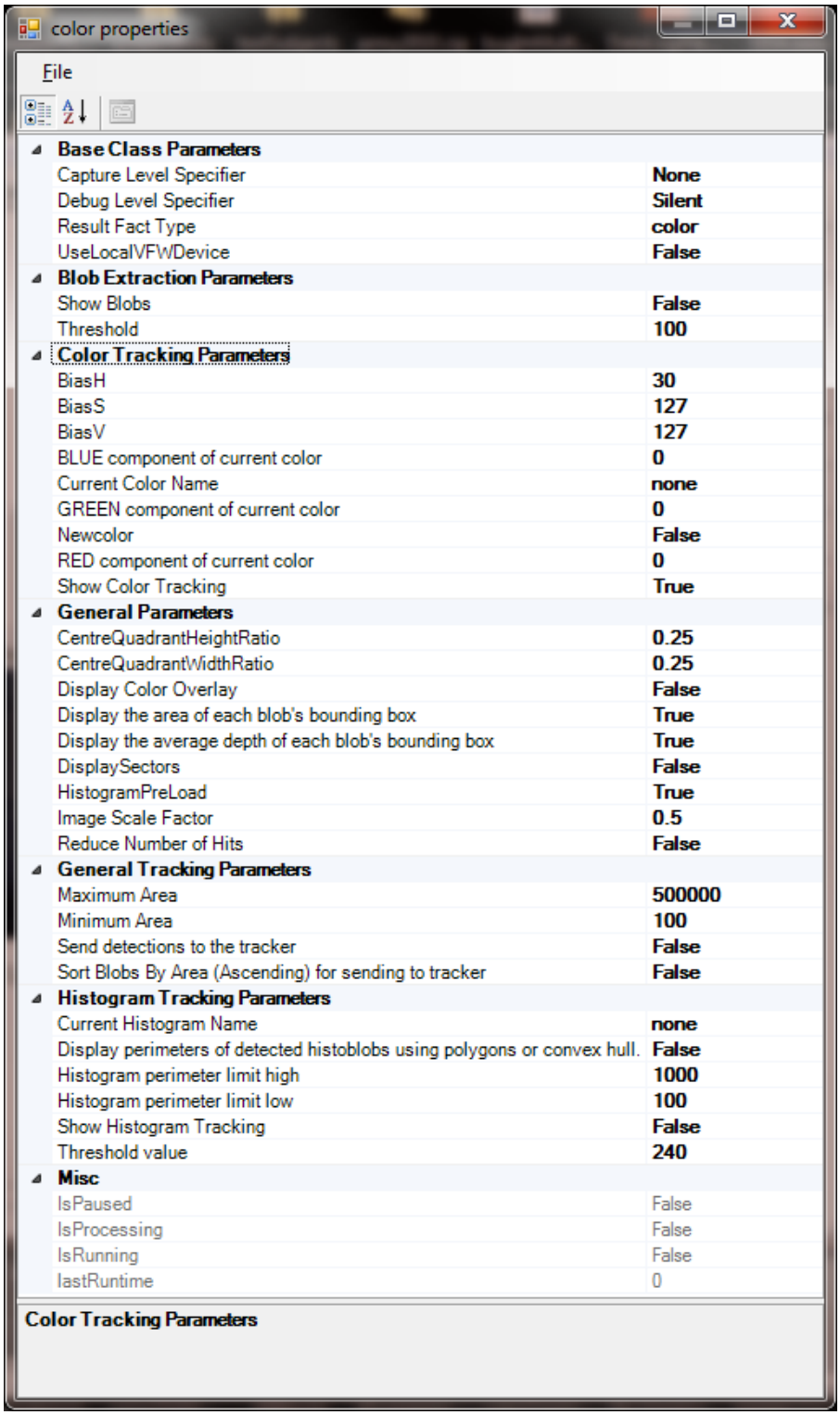


Fig. 141 Color Tracking subsim properties

Blob Extraction Parameters

Show Blobs

This parameter tells the subsim to detect and display all the blobs detected of any color on the screen. This is purely for diagnostic purposes (but is rather fun to watch).

Threshold

This is the parameter threshold for binary image calculation for color-blob detection. A value of 100 seems to work well.

Color Tracking Parameters

BiasH

This value is applied to the hue value of the designated color. If the hue value of the current pixel is the same as the designated hue plus or minus the BiasH value, the pixel is considered a match.

BiasS

This value is applied to the saturation value of the designated color. If the saturation value of the current pixel is the same as the designated saturation plus or minus the BiasS value, the pixel is considered a match.

BiasV

This value is applied to the brightness value of the designated color. If the brightness value of the current pixel is the same as the designated brightness plus or minus the BiasV value, the pixel is considered a match.

BLUE Component of Current Color

This value is B or blue component of the RGB representation of the current color.

GREEN Component of Current Color

This value is G or green component of the RGB representation of the current color.

RED Component of Current Color

This value is R or red component of the RGB representation of the current color.

Current Color Name

This value is the name of the color being detected. It is either the name of a color, histogram, or eyedropper.

Newcolor

This tells the single color detector (as opposed to histogram-based detection) that the eyedropper has been used and sets the current color to the color of the pixel selected. The name of the current color is also changed to eyedropper.

Show Color Tracking

This tells the single color detector to start detecting and displaying regions of color that correspond to the current color.

General Parameters

Display the Area of Each Blob's Bounding Box
When set to true, the area of each bounding box is displayed in the center of the box.

Display the Average Depth of Each Blob's Bounding Box
When set to true, the depth (distance to) of each bounding box is displayed in the center of the box above the area.

HistogramPreload
When set to true, the histogram data that are saved on disk will be loaded into each histogram fact as it is being constructed at initialization. If not true, the histograms will be loaded on demand.

Reduce Number of Hits
When set to true, overlapping bounding boxes will be combined and boxes wholly contained in other boxes will be eliminated.

General Tracking Parameters

Maximum Area
Specifies, in square pixels, the maximum size of a color blob that will be reported.

Minimum Area
Specifies, in square pixels, the minimum size of a color blob that will be reported.

Send Detections to the Ttracker
If true, detections will be sent to the Tracking subsim(s) depending on which are specified in the config file.

Sort Blobs by Area (ascending) for Sending to the Tracker
If true, detections will be sorted in order of descending area before being sent to the tracker(s).

Histogram Tracking Parameters

Current Histogram Name
Similar to the current color name, this variable is displayed in a status line across the top of the image as the color status line is when the debug level is on chatty.

Display Perimeters of Detected Histoblobs using Polygons or Convex Hull
This value determines whether detected histogram matches are cordoned off using polygons (true) or convex hull (false).

Histogram Perimeter Limit High
This value specifies the maximum permissible length in pixels of the perimeter of a histogram blob.

Histogram Perimeter Limit Low
This value specifies the minimum permissible length in pixels of the perimeter of a histogram blob.

Show Histogram Tracking
This tells the single color detector to start detecting and displaying regions of color that correspond to the current color.

Threshold Value
This parameter is the lowest value to consider as a cutoff for thresholding the image for histogram detection.

The File menu item contains the following commands (Fig. 142):

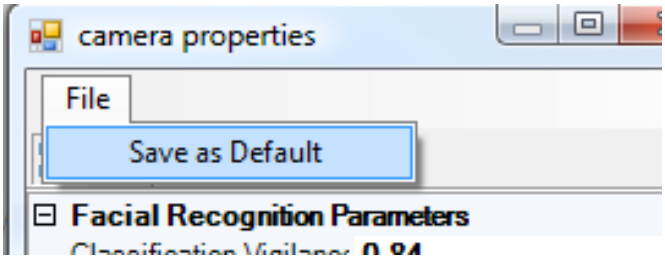


Fig. 142 Property page file menu item

Save as Default will write these values out to the application's config file (example), as follows:

```
      <add name="imagecorrelation"
type="Brain.CSubSimProcessorImageCorrelation" assembly="Brain"
run="false">
        <properties>
          <add property="QuickFind" value="False" />
          <add property="corrThreshold" value="-1" />
          <add property="fontSize" value="35.5" />
          <add property="PRELOADIMAGES" value="false" />
          <add property="ResultFactType" value="imagecorrelation" />
          <add property="DebugLevel" value="Silent" />
        </properties>
      </add>
```

Upon startup of the program, these values will be read in and assigned to the properties for that subsystem.

10. Traditional Neural Networks

The Network Builder tool allows the user to create a traditional feed forward neural network for use as a perceptual system within SS-RICS. Traditional neural

networks suffer from catastrophic forgetting. Catastrophic forgetting means that once a network is trained, for it to learn new information properly, it must be trained on all of the previous information as well as the new information. So, it is best to train them on sets of primitives that are not likely to change (letters in the alphabet) so that new information does not need to be added.

The topology of the network (the number nodes and layers between the nodes) are defined using the topology building tool (Fig. 143). The network being defined in this example consists of 3 layers with 180 inputs, 4 hidden layers, and 3 output layers. The output layer names are Room, Room2, and Hallway, which the user has defined.

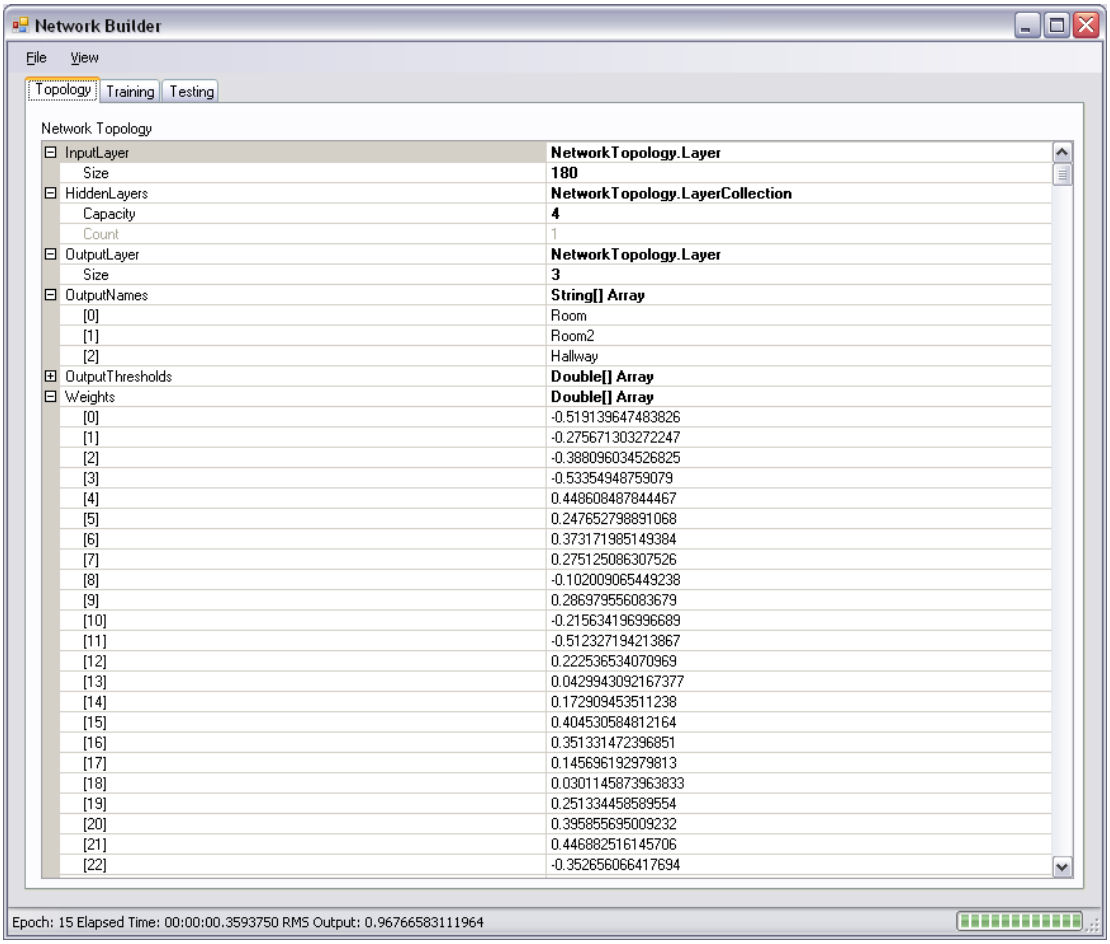


Fig. 143 Network Builder topology panel

The next tab of the tool contains the training section for the network builder. To enter training files, double-click on the files listing to add new files. The names of each category (in this case Room, Room2, and Hallway) can be only changed on the topology screen. Once the input files have been set, hit the Train button. If your number of inputs does not match the number you defined in the topology section,

you will get an error. You should see that the network converges, as in Fig. 144. If not, there is a problem with the network.

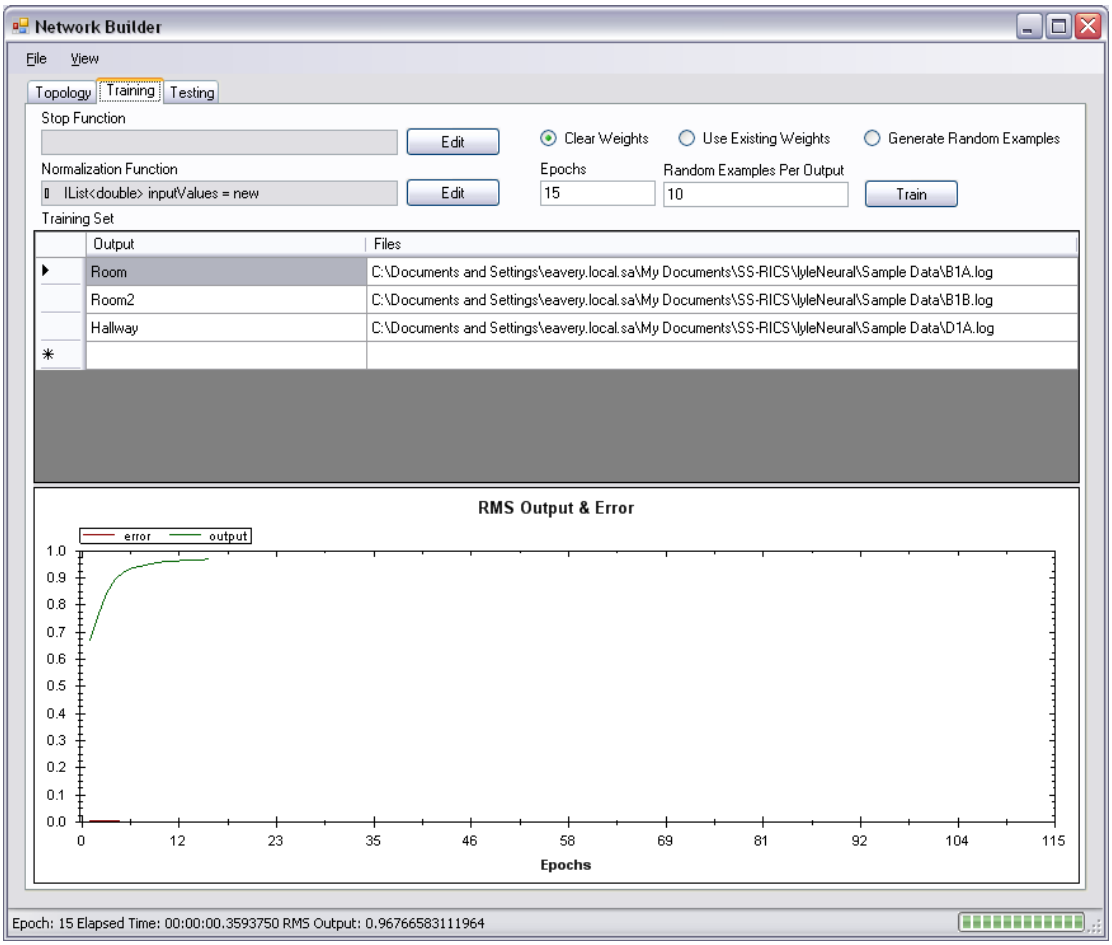


Fig. 144 Network Builder training interface

To check your trained network against outside data, go to the Testing tab. Here you can input a new file for testing against your network. You can test against one laser scan (input file line number) or a collection of scans by using the Use All Lines button. Once you have browsed to your testing network, you can hit the Test button to see the output of the network. The network will display the likely matches to your categories. In Fig. 145, the output screen has identified the current input of laser data has a hallway.

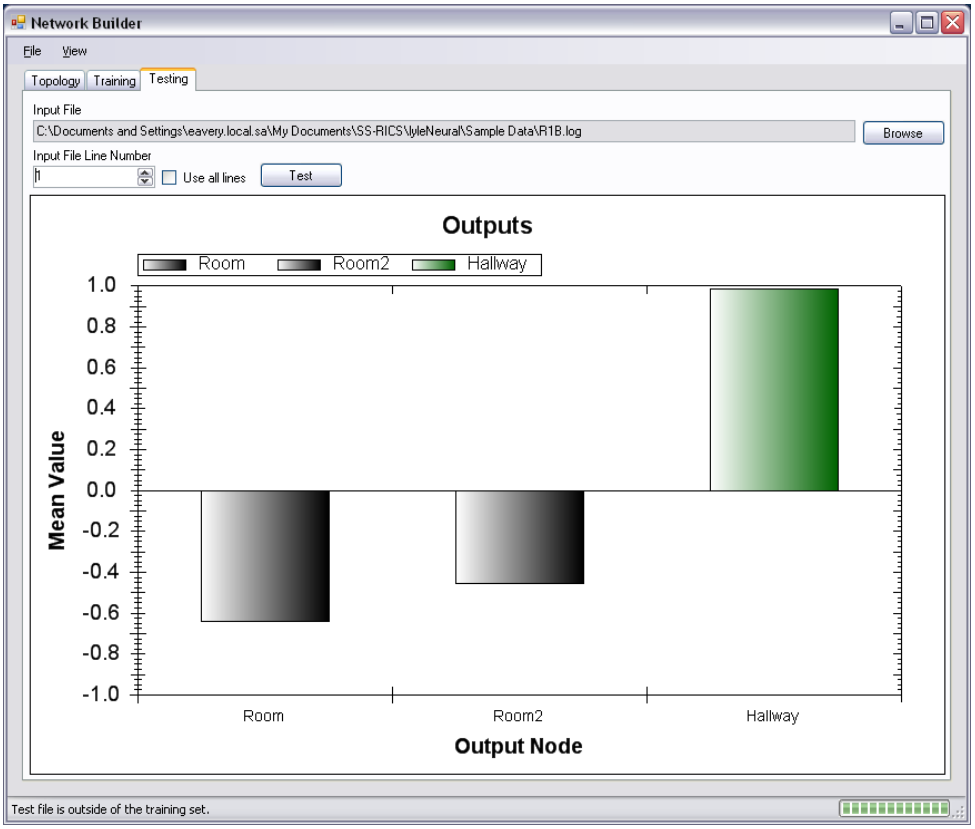


Fig. 145 Network Builder testing panel

11. Adaptive Resonance Theory (ART) Neural Networks

ART (Carpenter and Grossberg 2003) is a neural network that attempts to overcome problems with catastrophic forgetting. The catastrophic-forgetting problem happens when the modeler tries to add additional information into a network without first retraining the network on all of the old information. The old information will be lost and only the new information will remain. However, the ART network will allow a user to add additional information without having to retrain the entire network.

The ART neural network, Map Neural, is one of 2 neural networks implemented and in use on the SS-RICS robot. ARTMap is a designation given by the code author (Aaron Garrett) to distinguish the package from another called ART, which he also wrote. The ARTMap provides the facility to create a supervised neural network in which you inform the network as to what it is currently learning. The ART package (not implemented here) provides the tools to create an unsupervised network in which the package makes its own decision as to what it has learned. Since we require the ability to categorize the items it has learned, we use the ARTMap implementation.

11.1 Overview

The ARTMap takes as its input a vector of fixed and arbitrary length of real numbers whose values describe the salient features of an item you wish the ARTMap to learn. A very simple example would be the readings from the laser range device on the robot.

Say a laser range device on the robot takes 7 readings over an arc of 180°. These readings report the distance from the robot to whatever object is in the path of the laser as it sweeps. If the laser reports a distance of perhaps 1 m for each of the 7 readings (1,1,1,1,1,1,1), one could feed that to the ARTMap and ask it to categorize it as a wall the robot is staring straight at. If, on the other hand, it reported a distance of 1 m for the first and last 2 readings and a distance of 10 m for the middle values (1,1,10,10,10,1,1), this vector of values can be sent to the ARTMap, and one could instruct it to classify it as a wall with an opening in it that extends back 10 m, which could be called a Door or Doorway. Similarly, vector readings of different contours can be fed to the network and categorized as the user sees fit. Several sets of vectors with slight variations are required to teach the ARTMap a category.

The salient features of a laser scan are fairly straightforward, as they are purely distance related and provide a numerical representation of a 2-D shape you wish the ARTMap to ultimately come to learn and recognize. However, the ARTMap can be fed any set of data deemed representative of a feature/object you wish it to learn. Besides simple laser range data, values gleaned from images can also be used to teach it. An image in SS-RICS can be processed using openCV algorithms to obtain a reduced set of information from it such as edge lines or contour coefficients (called moments in openCV). These moments can be vectored into the ARTMap just as the laser data were and used to teach it what we consider to be the important aspects of a particular camera image. An image of a common desktop item like a stapler can be taught to the ARTMap and later classified as such when asking the ARTMap to identify it in an image.

11.2 ARTMap Topics

Using the ARTMap software encompasses the following items:

- Initializing a network
- Saving and loading a network
- Teaching the network a new category
- Evaluating a scene using the ARTMap/Vigilance

11.3 Initializing a Network

The ARTMap must be initialized (or loaded if it has already been initialized and saved) before it can be used. This amounts to telling it how many features are to be used in learning as well as classifying and setting the vigilance parameter (more on that in a bit). This is accomplished through a goal such as the following:

Execs (moments initartmap 7 .9)
This command tells the Moments subsim to initialize an ARTMap neural network with 7 features (input vector of 7 doubles) that will be used as learning input and classification input. It also tells it to use a vigilance factor of 0.9 when performing classifications.

The following is an example of an ARTMap being initialized for use with laser range data:

Execs (moments initartmap 180 .75))

This command tells the Moments subsim to initialize an ARTMap neural network with 180 features (input vector of 180 doubles) that will be used as learning input and as classification input. It also tells it to use a vigilance factor of 0.75 when performing classifications.

11.4 Saving and Loading a Network

Saving

An ARTMap MUST have been initialized *and* saved before it can be loaded. An ARTMap can be saved to disk as a binary data file containing all of the data it has accrued, if any. The following is example of a goal that saves the ARTMap for the Moments subsim out to disk:

Execs (moments saveartmap ArtMapNetwork))

When this goal is executed, an XML file named ARTMapTopologyMoments.xml is created, describing it and any trained categories. The name of the core file, ArtMapNetwork, is specified in the goal. An example of the resulting XML file is the following:

<ARTMapTopologyInfo>
<ARTMapCoreFilename>**ArtMapNetwork**</ARTMapCoreFilename>
<Vigilance>**0.9**</Vigilance>
<NumFeatures>**35**</NumFeatures>
<Classifications>
<Class0>**Stapler**</Class0>
<Class1>**Tape**</Class1>
<Class2>**Camera**</Class2>

```
  </Classifications>
</ARTMapTopologyInfo>
```

The name of the output file is specified ArtMapNetwork, the vigilance factor is 0.9, and the number of features is 35—all as described in the network's initialization. The following items have been taught to the ARTMap: Stapler, Tape, and Camera.

Similarly, the Points subsim ARTMap can be saved with the following goal:

Execs (points saveartmap sfam.art))

This can result in the following XML file (which, for the Points subsim, is named ARTMapTopologyLaser.XML). In this case, the name of the core file is sfam.art.

```
- <ARTMapTopologyInfo>
  <ARTMapCoreFilename>sfam.art</ARTMapCoreFilename>
  <Vigilance>0.75</Vigilance>
  <NumFeatures>180</NumFeatures>
- <Classifications>
    <Class0>ClosedDoor</Class0>
    <Class1>OpenDoor</Class1>
    <Class2>AjarDoor</Class2>
    <Class3>Box</Class3>
    <Class4>Hallway</Class4>
  </Classifications>
</ARTMapTopologyInfo>
```

Here we see that the name of the core file is sfam.art and that 5 different items have been taught to the ARTMap: ClosedDoor, OpenDoor, AjarDoor, Box, and Hallway.

You can create as many core files as you like, and they will be represented in the XML files.

Loading

An ARTMap can be loaded from disk as a binary data file containing all of the data it has accrued as defined by the XML file created when it was saved last. Typically, (and for now, exclusively) it is loaded using a goal like the following:

Execs (moments loadartmap))

Since this is the Points subsim, the interface will look for the file ARTMapTopology.xml in the top level of the SS-RICS directory and proceed to load the data from the information contained therein. Similarly, for the Moments subsim (the same goal but with the word points replaced with moments), the interface would look for a file named ARTMapTopologyMoments.xml and proceed to load from there.

11.5 Teaching the Network a New Category

After the ARTMap is initialized, it is ready to start learning. Teaching it a new category (and continuing its education with an existing category) is done, once again, through goals.

For the Points subsim, the data that get passed to the ARTMap are from the laser ranging device. When the robot's laser is scanning an area you wish the ARTMap to learn, the following goal can be executed to teach the ARTMap the current scene and tell it what it is supposed to be:

Execs (points trainartmap door))

In this case, the scene the user wishes the ARTMap to learn features a door. When executed, the goal will send the current vector of laser readings to the ARTMap. This goal will only send one vector. The following goal can be modified to send more than one reading:

Execs (points trainartmap door 1000))

This command will send the next 100 such vectors to the ARTMap in succession. Generally, the more exemplars a neural network is taught, the better it can classify.

For the Moments subsim, the data that gets passed to the ARTMap is from the current image from the camera and are the moments (as previously described) derived from the image using the openCV algorithms. The same goal shown for the Points subsim can be used to teach the ARTMap the current scene by simply changing the word points in the Exec consequent to moments.

Both the Moments and Points subsims can support their own ARTMap simultaneously.

11.6 Evaluating a Scene Using ARTMap/Vigilance

Once the ARTMap has been trained to recognize various artifacts observed with the laser range device or camera, it can perform an evaluation to see if anything it has been taught to recognize is visible in the current camera frame or laser sweeps. This is done continuously by the Moments and Points subsims (as long as an ARTMap has been initialized or loaded), which will send the current image or laser sweep data, respectively, to the ARTMap for evaluation. The ARTMap will compare the data with its store of learned categories and attempt to find a match. It will report its findings in the form of a fact which, for the moments, is named CurrentClassification of type ARTMap with a slot called Value, which will either be what the ARTMap considers to be a match on one of its known categories (like Stapler) or UNKNOWN if it could find no suitable match.

For the Points subsim, this fact has a unique name generated using the GUID utility, a type of ARTMapLocationType, and a slot called Location with the result of the classification from the ARTMap. This could be, for instance, AjarDoor or, as with the moments example, UNKNOWN, if no match is found.

The vigilance value weighs heavily in learning and classifying. A lower value produces more general and forgiving categories, allowing for a very lenient match, and commensurately higher values force a tighter, more rigid learning and matching scheme. A good balance has been found to be 0.75 for the Points subsim and 0.9 for the Moments subsim. A value of 0 would be useless, as everything would match the first category examined and a value of 9.99 would result in a match under the strictest of circumstances. (For instance, the data entered matched almost exactly the data already learned.)

12. Event Log Replay

The intent of Event Log Replay is to allow an operator the ability to control a robot and then have the robot replay those actions in sequence. The operator can also use this as the basis for creating new goals or actions for the robot without having to program all the actions in from scratch. The robot creates a log file of actions when the operator is controlling it. The robot can then read the log file when it is executing goals.

The brain interface keeps a log of all actions performed on the robot from the Motion subsim. This log (an example follows) contains the time a command was issued, the command itself, and duration of the command.

```
09:31.7  SpeedSetCommand          600
09:32.9  MotionCommand    turnright          0
09:39.3  MotionCommand    stop
09:42.4  GoalStarted      gTestGoal
09:44.9  GoalEnded        gTestGoal       2515
09:47.5  MotionCommand    moveforward        0
09:51.3  MotionCommand    stop
09:53.7  MotionCommand    turnright          0
09:57.0  MotionCommand    stop
09:58.3  MotionCommand    moveforward        0
10:01.9  MotionCommand    stop
10:04.8  GoalStarted      gTestGoal
10:07.2  GoalEnded        gTestGoal       2409
10:07.5  SpeedSetCommand          594
```

Implicit in the data are the durations of each command, which are simply the difference between the time of the command and the time of the next command.

To select a log file to replay, select the option as follows in Fig. 146:

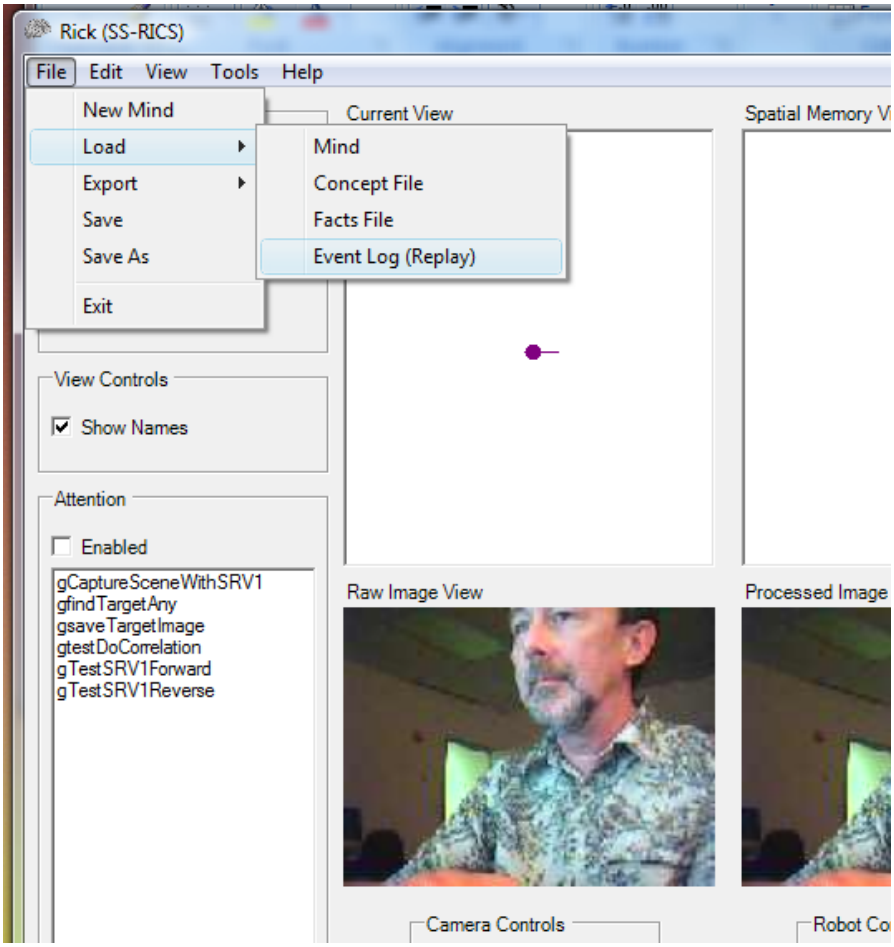


Fig. 146 Event Log Replay command menu item

This will give you a file dialog menu from which to select the log file you wish to replay. It is typically named something like Events.csv. Note that the file type is .csv, which is a comma-separated file suitable for input into Excel.

The system will then repeat all commands in the file in the same sequence and with the same duration as originally performed.

When the Event Log Replay has finished, it will display a box like the one shown in Fig. 147.

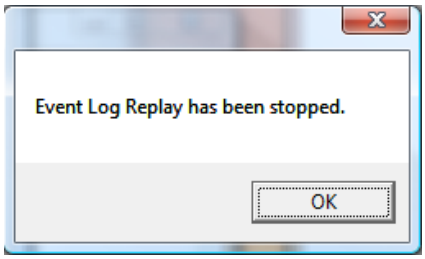


Fig. 147 End of replay notification

Click OK to continue processing as normal.

13. Using the Simulator

SS-RICS will work with any simulator that simulates the Pioneer robot API via the ARIA command interface (Fig. 148). SS-RICS will automatically launch the Mobile Robots MobileSim if it has been installed on the local system.

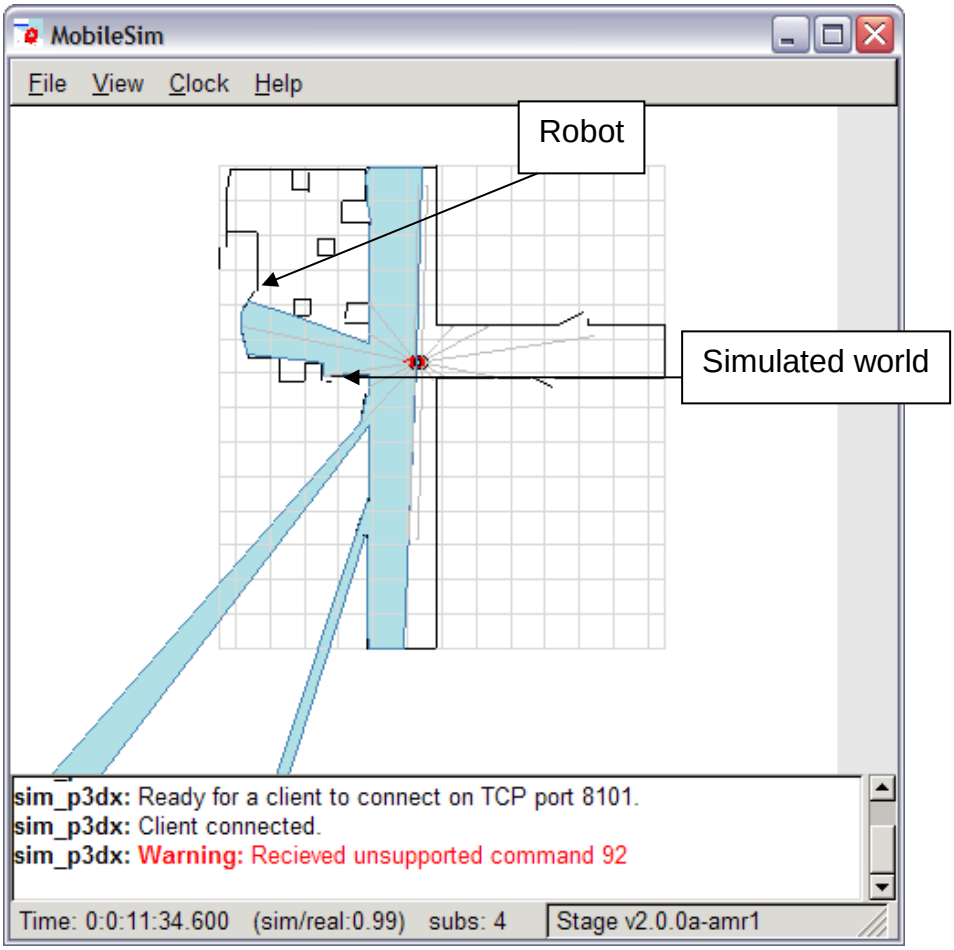


Fig. 148 Robot simulator display

What Is MobileSim

MobileSim is software for simulating MobileRobots platforms and their environments and for debugging and experimentation with ARIA. It replaces SRIsim previously distributed with ARIA. MobileSim builds upon the Stage simulator (created by Richard Vaughan, Andrew Howard, and others) as part of the Player/Stage project, with some modification by ActivMedia. This is an early beta release of MobileSim, and some features are still in development. MobileSim uses line data from a MobileRobots map (.map) file generated by Mapper3 or Mapper3-Basic to simulate walls and other obstacles in the environment.

More information can be found at Mobile Robots web site:
http://robots.mobilerobots.com/.

14. Configuration File

The initial configuration of SS-RICS and its Brain Interface are controlled by the values present in the XML-based BrainInterface.exe.config file. A sample .config file follows:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="SubSimProcessors" type="Brain.Config.SubSimProcessorConfigSection, Brain"
/>
    <section name="Activations" type="Brain.Config.MemoryActivationConfigSection, Brain" />
  </configSections>
  <SubSimProcessors>
    <ProcessorConfigurations>
      <add name="motion" type="Brain.CSubSimProcessorMotion" assembly="Brain" run="true" />
      <add name="voice" type="Brain.CSubSimProcessorVoice" assembly="Brain" run="true" />
      <add name="hearing" type="Brain.CSubSimProcessorHearing" assembly="Brain" run="true" />
      <add name="choice" type="Brain.CSubSimProcessorChoice" assembly="Brain" run="true" />
      <!--<add name="faces" type="Brain.CSubSimProcessorFace" assembly="Brain" run="true" />
      <add name="templatematch" type="Brain.CSubSimProcessorTemplateMatch" assembly="Brain"
run="true" />
      <add name="blobs" type="Brain.CSubSimProcessorBlobs" assembly="Brain" run="true" />
      <add name="moments" type="Brain.CSubSimProcessorMoments" assembly="Brain" run="true" />
      <add name="imagecorrelation" type="Brain.CSubSimProcessorImageCorrelation"
assembly="Brain" run="true" />
      -->
      <add name="points" type="Brain.CSubSimProcessorPoints" assembly="Brain" run="true" />
      <add name="linesegment" type="Brain.CSubSimProcessorLineSegment" assembly="Brain"
run="true" input="Points">
        <properties>
          <add property="lineFinderMaxAveDistFromLine" value="20" />
          <add property="lineFinderMaxDistFromLine" value="30" />
          <add property="lineFinderMinLineLength" value="75" />
          <add property="lineFinderMinPointsInLine" value="3" />
          <add property="lineFinderLinesCloseEnough" value="75" />
          <add property="lineFinderAngleTol" value="30" />
          <add property="lineFinderMinLinePoints" value="2" />
          <add property="lineFinderMinLineLen" value="40" />
        </properties>
      </add>
      <add name="line" type="Brain.CSubSimProcessorLine" assembly="Brain" run="true"
input="LineSegment" />
      <add name="intersect" type="Brain.CSubSimProcessorIntersect" assembly="Brain"
run="true" input="Line" />
      <add name="corner" type="Brain.CSubSimProcessorCorner" assembly="Brain" run="true"
input="Intersect" />
      <add name="gap" type="Brain.CSubSimProcessorGap" assembly="Brain" run="true"
input="Line" />
      <add name="region" type="Brain.CSubSimProcessorRegion" assembly="Brain" run="true"
input="Corner,Gap" />
    </ProcessorConfigurations>
  </SubSimProcessors>
  <Activations>
    <ActivationParameters>
```

```
      <add name="color" basestrength="1" decaystrength="0.29" returntobasestrength="5000"
deletethreshold="0.01" baselinenoisesigma="0.5" activationnoisesigma="0.7" decays="False" />
    </ActivationParameters>
  </Activations>
  <appSettings>
    <!--   User application and configured property settings go here.-->
    <!--   Example: <add key="settingName" value="settingValue"/> -->
    <!--
                  Below are example options for configuring the brain. You may specify some or
all of the options.
                  For options that specify a file or collections of files you may specify the
full or absolute
                  paths delimited by commas.

                  <add key="Mind" value="TestMind.dat" />

                  <add key="Connection" value="Serial" />

                  <add key="Connection" value="Simulator" />
                  <add key="Map" value="Maps\Office.map" />

                  <add key="Connection" value="Network" />
                  <add key="IPAddress" value="192.168.2.7" />
                  <add key="RobotPort" value="8101" />
                  <add key="LaserPort" value="8102" />
                  <add key="UseStereoCamera" value="true" />
                  <add key="StereoCameraPort" value="8103" />
                  <add key="UsePTZCamera" value="true" />
                  <add key="PTZCameraPort" value="8104" />

                  <add key="Facts" value="Facts Test.txt,C:\MyFacts.txt" />
                  <add key="Goals" value="SceneTest.rbx,gWanderAround.goal" />
                  <add key="Concepts" value="Building Concepts.txt,World.txt" />
                  <add key="EnableAttention" value="true" />
                  <add key="AutoOpenGoalEdit" value="true" />
                  <add key="Goals" value="Goals\gTestlog.goal" />


    <add key="hpUsersManual.HelpNamespace" value="(None)" />
    <add key="RobotType" value="Packbot"/>
    <add key="Connection" value="network" />
    <add key="IPAddress" value="192.168.75.2" />
    <add key="RobotPort" value="14171" />
    <add key="LaserPort" value="8101" />
    -->
    <add key="RobotType" value="Pioneer" />
    <add key="Connection" value="Simulator" />
    <add key="Map" value="Maps\Casel-EA.map" />
    <!--<add key="UsePTZCamera" value="true" />
    <add key="PTZCameraType" value="AVI File" />-->
  </appSettings>
</configuration>
```

There are 4 main sections to the .config file, as follows:

- ConfigSections
- SubSimProcessors
- Activations
- Application Settings

14.1 ConfigSections

This section lists the components that comprise the ConfigSections, SubSimProcessors, and Activations.

14.2 SubSimProcessors

In the parameter key/value pair, `name="line"` (an example), which specifies the name of the subsim, the value *must* be lowercase. In turn, if this subsim feeds

another (like the Intersect subsim), the name in the `input="line"` key/value pair must also be lowercase.

At the time of this writing, there are 25 subsims: Boredom, Visual Boredom, Faces, Choice, Corner, Gap, Hallway, Hearing, Intersect, Laser, Line, Linesegment, Motion, Motion Detection, Object, Parallel, Points, Question, Region, Relation, Sequence, Sonar, Spacial, Voice, and Wall.

They are made visible to the user in the Brain Interface Subsim Config dialog box, which can be accessed by choosing the following menu item from the Brain Interface main panel (Fig. 149).

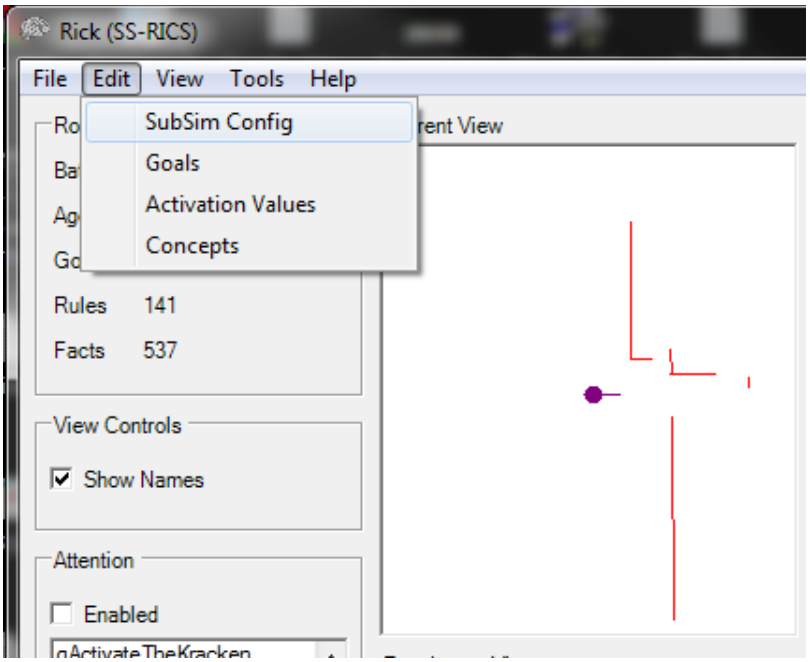


Fig. 149 Subsim configuration panel, open menu item

This brings up this dialog box shown in Fig. 150.

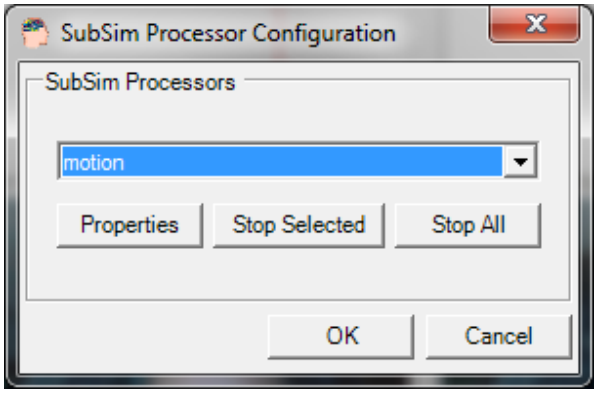


Fig. 150 Mind configuration panel showing drop-down of subsims

The subsims listed in this box are only the ones specified (not commented out) in the sample config file. The following line,

```
<add name="motion" type="Brain.CSubSimProcessorMotion" assembly="Brain"
run="true" />
```

for instance, tells the Brain Interface to load the subsim "motion" and to start it up. The subsim could have been marked `run="false"` and loaded but not started. It can subsequently be started by the user by using the controls under the drop-down menu (Fig. 151).

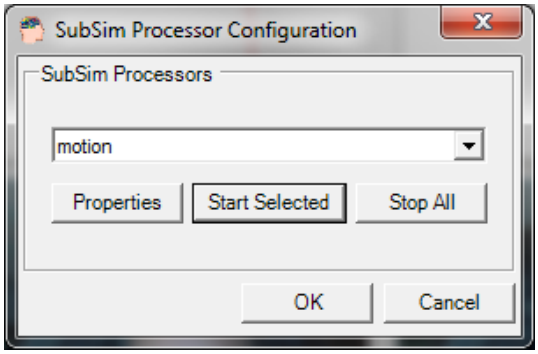


Fig. 151 Button control to start selected subsim

Some subsims feed data to other subsims like the Points subsim, which feeds points detected by the laser to the LineSegment subsim, which feeds line segments (derived from the laser points) to the Line subsim, which forms complete lines from the line segments sent to it, as shown in the following:

```
      <add name="points" type="Brain.CSubSimProcessorPoints"
assembly="Brain" run="true"/>
      <add name="linesegment" type="Brain.CSubSimProcessorLineSegment"
assembly="Brain" run="true" input="points"/>
      <add name="line" type="Brain.CSubSimProcessorLine" assembly="Brain"
run="true" input="lineSegment"/>
```

The parameter key/value pair `assembly="Brain"` indicates which .dll the subsim is present in. In all cases it is the Brain.

Within the SubSimProcessor section, the properties that can be set via the properties control panel described earlier can be saved for reload upon startup of SS-RICS. When you select to save the values with the file menu on the properties panel, the values are written to the .config file. An example is the LineSegment subsim in the config file just shown:

```
      <add name="linesegment" type="Brain.CSubSimProcessorLineSegment"
assembly="Brain" run="true" input="Points">
        <properties>
          <add property="lineFinderMaxAveDistFromLine" value="20" />
          <add property="lineFinderMaxDistFromLine" value="30" />
          <add property="lineFinderMinLineLength" value="75" />
          <add property="lineFinderMinPointsInLine" value="3" />
```

```
        <add property="lineFinderLinesCloseEnough" value="75" />
        <add property="lineFinderAngleTol" value="30" />
        <add property="lineFinderMinLinePoints" value="2" />
        <add property="lineFinderMinLineLen" value="40" />
      </properties>
    </add>
```

14.3 Activations

The parameters that govern the activation aspect of SS-RICS can be persisted, for each memory type, in the configuration file as well. For example, the color memory type has its activation values written to the file, as shown in the following:

```
  <Activations>
    <ActivationParameters>
      <add name="color" basestrength="1" decaystrength="0.29"
returntobasestrength="5000" deletethreshold="0.01"
baselinenoisesigma="0.5" activationnoisesigma="0.7" decays="False" />
    </ActivationParameters>
  </Activations>
```

14.4 Application Settings

Many aspects of the application can be controlled via the applications setting section of the .config file. These include the following:

- Opening the Goal Edit dialog box upon startup:

  ```
  <add key="AutoOpenGoalEdit" value="true" />
  ```

- Specifying a particular Mind file to load:

  ```
  <add key="Mind" value="TestMind.dat" />
  ```

- Specifying a particular Map file to load:

  ```
  <add key="Map" value="Maps\Office.map" />
  ```

- Selecting the PTZ camera and which port to connect to it on:

  ```
  <add key="UsePTZCamera" value="true" />
  <add key="PTZCameraPort" value="8104" />
  <add key="PTZCameraType" value="USARSIM (Network)" />
  ```

- Selecting the network connection mode and which IP, robot port, and laser port to access:

  ```
  <add key="Connection" value="Network" />
  <add key="IPAddress" value="192.168.100.100" />
  <add key="RobotPort" value="8101" />
  <add key="LaserPort" value="8102" />
  ```

- Selecting simulator mode:

  ```
  <add key="Connection" value="Simulator" />
  ```

- Selecting which Facts file to load:

  ```
  <add key="Facts" value="Facts Test.txt,C:\MyFacts.txt" />
  ```

- Selecting which Goals file to load:

```
<add key="Goals" value="Goals\gTestlog.goal" />
```

- Selecting which Concept (ConceptNet) file to load:

```
<add key="Concepts" value="Building Concepts.txt,World.txt" />
```

- Selecting the type of robot (SS-RICS can access several types):

```
<add key="RobotType" value="Pioneer" />
```

- Specifying that the Attention feature be activated:

```
<add key="EnableAttention" value="true" />
```

- Automatically delete goals existing in the mind that are the same as other goals being loaded without a prompt:

```
<add key="AutoDeleteExistingGoals" value="True"/>
```

- Automatically save goals existing in the mind when the robot is shut down:

```
<add key="AutoSaveGoals" value="True"/>
```

- Automatically run a goal upon SS-RICS startup:

```
<add key="AutoRunGoal" value="gExploreWorld"/>
```

- Autoload a mind file:

```
<add key="Mind" value="TestMind.dat" />
```

- Supress audit trail of goals that are executed:

```
<add key="SupressGoalDebugOutput" value="True"/>
```

- Select the type of simulator to use:

```
<add key="SimulatorType" value="mobilesim"/>
```

- Specify the IP address of the camera host:

```
<add key="CameraHost" value="192.168.100.101"/>
```

- Direct the robot to use a stereo camera:

```
<add key="UseStereoCamera" value="true"/>
```

- Port to use for the stereo camera if remote:

```
<add key="StereoCameraPort" value="1803"/>
```

- Type of stereo camera to use:

```
<add key="StereoCameraType" value="Bumblebee (Firewire)"/>
```

- Use the voice on the robot:

```
<add key="UseRemoteVoice" value="true"/>
```

- Port for robot remote voice:

```
<add key="RemoteVoicePort" value="8108"/>
```

15. Command Line Switches

The Brain Interface can be started with several options available through command line switches. There are currently 5 such switches available, as follows:

- Autoload: tells the braininterface to initialize the brain with the mind file specified in the config file.
- Config: tells the braininterface that the next argument in the command line is the name of the configuration file it is to use.
- Skipbuildupdate: tells the braininterface to not check for a new build available through Team Foundation Server.
- Showconsole: tells the braininterface to display the DOS console.
- Debug: sets the robot up for remote debugging.

None of the command line switches are mutually incompatible. An example of usage follows:

braininterface.exe /autoload /config braininterface.config /skipbuildupdate /showconsole

16. How to Get Help with SS-RICS

You must request an account to access SS-RICS releases and source code. Each request for help with specific questions or concerns will be evaluated, and you should receive a response within 24 h. Contact either of the following:

Troy Kelley: troy.d.kelley6.civ@mail.mil

Sean McGhee: sean.m.mcghee.ctr@mail.mil

17. References

Anderson JR, Lebiere C. The atomic components of thought. Mahwah (NJ): Lawrence Erlbaum Associates; 1998.

Avery E, Kelley TD, Davani D. Using cognitive architectures to improve robot control: integrating production systems, semantic networks, and sub-symbolic processing. Proceedings of the 15th Annual Conference on Behavioral Representation in Modeling and Simulation (BRIMS); 2006 May 15–18; Baltimore, MD. p. 25–33.

Busemeyer JR, Johnson JG. Micro-process models of decision-making. In: Sun R, editor. Cambridge handbook of computational cognitive modeling. Cambridge (UK): Cambridge University Press; 2006.

Carpenter GA, Grossberg S. Adaptive resonance theory. In: Arbib MA, editor. The handbook of brain theory and neural networks. 2nd ed. Cambridge (MA): MIT Press; 2003. p. 87–90.

Kalal Z, Mikolajczyk K, Matas J. Tracking-learning-detection. IEEE Tran on Pattern Analysis and Machine Intelligence. 2010;6(1):1–14.

Kelley TD. Symbolic and sub-symbolic representations in computational models of human cognition: what can be learned from biology? Theory and Psychology. 2003;13:6.

Kelley TD. Developing a psychologically inspired cognitive architecture for robotic control: the symbolic and sub-symbolic robotic intelligence control system (SS-RICS). International Journal of Advanced Robotic Systems. 2006;3:3.

Kelley TD. Using a cognitive architecture to solve simultaneous localization and mapping (SLAM) problems. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2006. Report No.: ARL-MR-0639.

Liu H, Singh P ConceptNet – a practical commonsense reasoning tool-kit. BT Technology Journal. 2004;22(4):211–226.

Newell A. Unified theories of cognition and the role of Soar. In: Michon J, Akyurek A, editor. Soar: a cognitive architecture in perspective. Cambridge (MA): Kluwer Academic; 1992.

INTENTIONALLY LEFT BLANK.

Appendix. Goal and Rule Snippets

This appendix appears in its original form, without editorial change.

Goal and Rule Snippets

Goals and rules are the primary motivating force behind the behavior of SS-RICS. Whereas the robot is capable of sensing it's environment without goals and rules, it is quite limited in what it can do with that information without any actual codified strategy or purpose. Goals and Rules allow the user to manipulate the sensory input, save and retrieve and process information stored in the working directory subfolders such as laser scans (location and environment data), tld (predator tracking) files, and image files (for template matching; image correlation). Following are examples of creating and manipulating these resources.

- Fetch the date of an image whose activation is currently the highest (activation values are explained in the Activation section):

This should return the most highly activated image in the realworld folder which we just created in the wakeup call. We concatenate the year, month, and day together and set the result as the date. This will be our top level episodic directory. Note we have to put in the seconds so that if we restart on the same day we will not keep writing to the same directory.

```
goal GetDate
	{
	rule GetDate((name? realworld/episodes/images *=*))
	{
		Set(Position State Event=Unknown)
		Set(Position State Seconds=?name.seconds)
		Set(Position State Day=?name.day)
		Set(Position State Month=?name.month)
		Set(Position State Year=?name.year)
		Set(Position State
Date=?name.year_?name.month_?name.day_?name.seconds_)
		Print(Setting Date to
?name.year_?name.month_?name.day_?name.seconds_)
		Quit(true)
	}
	rule DefaultgGetDate()
	{
		Quit(true)
	}
}
```

- Parse the comma delimited array of elements in the CurrentView/State fact in slot "movingFacts" – this is a list of regions found by the motion detect subsim and are labeled bogeyXX where XX is the numbering of the element:

First, the CurrentView/State fact is matched to assure it has a slot named "movingFacts". Then a fact called BogeyCount/Arg is added/set such that its slot "Value" has the number (count) of items in the comma delimited string contained in that slot. Next, the value of the first element (0 as specified by movingFacts[0]) will be printed Finally the goal will stop executing when the Quit directive is executed.

```
goal ParseAnArray
 {
       rule DoStuff((CurrentView State movingFacts=* *=*))
       {
       Set(BogeyCount Arg Value=?CurrentView.movingFacts[count])
       Print(first item "?CurrentView.movingFacts[0]_")
       Quit(true)
    }
 }
```

- Execute a goal from another goal.

To set off a chain of goal executions, the ability to execute a goal from a goal is useful. The next example executes the goals described in the previous 2 examples (note the names of each goal are prefixed with a lower-case “g” and that the rule will not fire unless the antecedent match is made on the CurrentView/State fact with slot movingFacts). The results will be reported in facts as described above and the goal StartTwoGoals will end when the Quit directive is executed:

```
goal StartTwoGoals
 {
       rule DoTwoGoals((CurrentView State movingFacts=* *=*))
       {
       Exec(Goal gParseAnArray)
       Exec(Goal gGetDate)
       Quit(true)
       }
 }
```

- Use the color subsim to find a color in a range.

This goal assumes the „brown“ subsim is running (the name given to an instance of the colortracking subsim) and the moments subsim is running. Next it matches on the brown/State fact created/maintained by the brown colortracking subsim and the momentsDepth/State fact created/maintained by the moments subsim. In each case, the assumption is that a stereo camera is in use to provide depth data. Next the brown subsim is directed to enter autocycle mode (see the colortracking section for details) followed by a command to search for all shades of brown available in the color facts (designated by brown*). After that has been accomplished the value of the closest shade of brown found is placed in the depth slot of the brown/State fact and printed out. Following that, the distance to the closest „moment“ (in millimeters) found by the moments subsim is accessed and printed. The user presumes the 2 depth values are correlated and after comparing the values either in the print window or with another goal, can make a decision as to the robot's next course of action.

```
 goal FindBrown
       {
       rule FindBrown((brown State depth=* *=*)
                   &&(momentsDepth State closemm=* *=*))
              {
              Execs(brown autocycleon)
              Execs(brown findcolor brown*)
```

```
            Print(?brown.depth)
            Print(?momentsDepth.closemm)
            }
      }
```

- Execute a loop with a counter.

This goal will loop around checking the Hardware/State fact to see if the camera is connected camera at 5000 ms intervals. It does one check and reports, after that the count variable is set to 2 and it will check but not report (this keeps it from reporting connected over and over). As soon as the camera state is false, it will report. Note the name of the goal is reflected in the Slots Attribute line above the goal. These particular slots when concatenated are the name of the goal. In this case, Check, Camera, and Now.

```
[Slots(verb=Check noun=Camera adverb=Now)]
goal CheckCameraNow
{
      rule InitializegCheckCameraNow()
            {
                  Print(initialize camera check)
                  Set(CameraCounter Arg Value=1)
                  DisableRule(Self)
            }
      rule CheckCameraNowOn((Hardware State CameraConnected=True *=*) &&
      (CameraCount Arg Value=2 *=*))
            {
                  Wait(5000)
            }
      rule CheckCameraNowTrue((Hardware State *=*;1
      CameraConnected=True;1)&& (CameraCount Arg Value=1 *=*))
      {
            Execs (Voice Say "My camera is connected")
            Set(CameraCount Arg Value=2)
            Wait(5000)
      }
      rule CheckCameraNowFalse((Hardware State *=*;1
CameraConnected=False;1))
      {
            Print(Warning camera is not connected)
            Execs (Voice Say "Warning My camera is not connected")
            Quit (true)
      }
}
```

- Proceed forward until the distance from the destination (a range depth) is less than 1000 millimeters (assumes the robot is moving forward).

This goal will continuously check to see if the robot is closing in on its destination (when a forward range distance reading is less than 1 meter as reported by the laser).

```
/* here is a strategy, this could be done in a sub sim, but this is simple enough
to do here.  This is also only for a sub goal of getting
to the front of the building.  Also we are setting a distance and an action value,
for the less than 1000 we are setting it to stop */
goal GetRangeFront
      {
      rule GetRangeFrontHigh((CurrentView? State RangeFront>1000 *=*))
            {
```

```
                Print(Getting range value high)

                /* we have to get our previous range value to use for
                determining reinforcement */
                Set(PreviousRangeFront Arg Value=?CurrentView.RangeFront)
                Print(?CurrentView.RangeFront)
                Set(Action Arg Value=MoveForward)
                Set(Distance Arg Value=1)

                Quit(true)
                }
        rule GetRangeFrontLow((CurrentView State RangeFront<1000 *=*))
                {
                Print(Getting range value low)
                Set(Action Arg Value=Stop)
                Set(Distance Arg Value=0)

                Quit(true)
                }
}
```

- Use the FaceRecognition subsim.

These goals will demonstrate how to initialise, train, save, load, retrain, update, and allow for the reinitialising of the subsim.

-init and train model in one goal. Model is Eigen with all default parameters. Training file is called train.dat and the command trainmodel uses that file to train the model. After goal is complete, the model is ready to classify a candidate face in the camera view.

```
goal QuickStartEigenTrain
{
        rule QuickStartEigenTrain()
        {
                Execs (eigenface setmodel Eigen none)
                Execs (eigenface settrainingfilename train.dat)
                Execs (eigenface trainmodel)
                Quit (true)
        }
}
```

-change the confidence threshold of the model to 1600. Allows the model to emit a prediction as to who the candidate belongs to as long as it does not exceed 16000 (a distance metric to nearest neighbor).

```
goal ChangeEigenThreshold
{
        rule ChangeEigenThreshold()
        {
                Execs (eigenface changethreshold 16000)
                Quit (true)
        }
}
```

-save a trained model to a file for later reload. The file is saved with the extension .yaml.

```
goal SetEigenSaveFile
{
        rule SetEigenSaveFile()
        {
                Execs (eigenface setsavefile eigen.yaml)
                Execs (eigenface savemodel)
                Quit (true)
        }
```

```
}
-or-
goal SetEigenSaveFile
{
	rule SetEigenSaveFile()
	{
		Execs (eigenface savemodel eigen.yaml)
		Quit (true)
	}
}
```

-load a trained model from a file from an earlier save. This saves a lot of time because training can take a few minutes depending on the size of the training base. The file to be loaded will have been saved with the extension .yaml.

```
goal SetEigenLoadFile
{
	rule SetEigenLoadFile()
	{
		Execs (eigenface setloadfile eigen.yaml)
		Execs (eigenface loadmodel)
		Quit (true)
	}
}
-or-
goal SetEigenLoadFile
{
	rule SetEigenLoadFile()
	{
		Execs (eigenface loadmodel eigen.yaml)
		Quit (true)
	}
}
```

-load a trained model from a file from an earlier save. This saves a lot of time because training can take a few minutes depending on the size of the training base. The file to be loaded will have been saved with the extension .yaml.

```
goal SetEigenLoadFile
{
	rule SetEigenLoadFile()
	{
		Execs (eigenface setloadfile eigen.yaml)
		Execs (eigenface loadmodel)
		Quit (true)
	}
}
```

-A simple goal to detect and report whether or not the laser is connected.

```
[Slots(verb=Check noun=My adverb=Laser)]
goal CheckMyLaser
{
	rule CheckMyLaser((Hardware State *=*;1 LaserConnected=True;1))
	{
		Execs (Voice Say "My laser is connected")
		Quit (true)
	}
	rule CheckMyLaserNot((Hardware State *=*;1
LaserConnected=False;1))
	{
		Execs (Voice Say "My laser is not connected")
		Quit (true)
	}
}
```

-A simple goal to retrieve and print selected average depths of screen sectors as computed in the colortracking subsim. The CurrentView/State fact is matched on and if it contains the slots ColorCenter, ColorBottom, ColorTop, ColorLeft, and ColorRight, the rule will fire after waiting 800 milliseconds, printing out the ColorTop, ColorCenter, and ColorBottom average depth values.

```
goal TestColorGet
{
	rule TestColorGet((CurrentView State ColorCenter=* ColorBottom=*
	ColorTop=*   ColorLeft=* ColorRight=* *=*))
	{
		Wait(800)
		Print (Top ?CurrentView.ColorTop_ Center ?CurrentView.ColorCenter_
		Bottom   ?CurrentView.ColorBottom_)
	}
}
```

-A set of goals that will use various capabilities of the tracking subsim.

These goals will demonstrate how to capture a face, send it to the tracker, and perform various functions regarding the tracked face.

-A goal which tells the face detection subsim to send a detected face to the tracking subsim (tracking subsim must have „faces“ as an input in the config file).

```
goal SendFaceToTracker
{
	rule SendFaceToTracker ()
	{
		Execs (faces trackface)
		Quit( True )
	}
}
```

-A goal which renames the face received from the faces subsim to „SEAN“
and saves it in the current working directory in a file called SEAN.tld.

```
goal TrackingRename
{
	rule TrackingRename()
	{
		Execs (tracking rename face1 SEAN)
		Quit( True )
	}
 }
```

-A goal which loads the current target file SEAN.tld from the current working directory.

```
goal TrackingLoadTargets
 {
	rule LoadTargets()
	{
		Execs(tracking loadtargets SEAN)
		Quit( True )
	}
 }
```

-A goal which ends tracking of the targets SEAN, ERIC, and TROY.

```
goal TrackingShutDown
 {
	rule TrackingShutDown()
	{
		Execs(tracking shutdown SEAN ERIC TROY)
		Quit( True )
	}
 }
```

-A goal which suspends tracking of the specified targets SEAN, ERIC, and TROY.

```
goal TrackingHibernate
 {
       rule TrackingHibernate()
       {
              Execs(tracking hibernate SEAN ERIC TROY)
              Quit( True )
       }
 }
```

-A goal which resumes tracking of the specified targets SEAN, ERIC, and TROY.

```
 goal TrackingWakeUp
 {
       rule TrackingWakeUp()
       {
              Execs(tracking wakeup SEAN ERIC TROY)
              Quit( True )
       }
 }
```

-A goal which forces training of the specified targets SEAN, ERIC, and TROY.

```
goal TrackingForceTraining
 {
       rule TrackingForceTraining ()
       {
              Execs (tracking forcetrain SEAN ERIC TROY)
              Quit( True )
       }
 }
```

-A goal which ceases forced training of the specified targets SEAN, ERIC, and TROY.

```
goal TrackingCeaseTraining
 {
       rule TrackingCeaseTraining()
       {
              Execs (tracking ceasetrain SEAN ERIC TROY)
              Quit( True )
       }
 }
```

-A goal which displays the tracking features used for the specified targets SEAN, ERIC, and TROY.

```
goal TrackingShowPoints
 {
       rule TrackingShowPoints()
       {
              Execs (tracking showpoints)
              Quit( True )
       }
 }
```

-A goal which stops display of the tracking features used for the specified targets SEAN, ERIC, and TROY.

```
goal TrackingHidePoints
 {
       rule TrackingHidePoints()
       {
              Execs (tracking hidepoints)
              Quit( True )
       }
 }
```

-A goal which limits the number of regions the tracker will attempt to track – 5 is usual – the more tracked the slower the update and higher the memory usage.

```
goal TrackingMaxTrackers
 {
       rule TrackingMaxTrackers()
       {
              Execs(tracking maxtrackers 5)
              Quit( True )
       }
 }
```

-A goal which specifies which designator to use to denote the tracked region – in this case a convex hull perimeter.

```
goal TrackingConvexhull
{
	rule TrackingConvexhull()
	{
		Execs (tracking selectdisplaytype convexhull)
		Quit( True )
	}
}
```

-A goal which deletes the tracking file tlds/faces/sean.tld. This object was saved to a subdirectory, tld/faces/, of the working directory and was named „tlds/faces/sean“ in a previous rename operation.

```
goal TrackingDeleteTLDS
{
	rule TrackingDeleteTLDS()
	{
		Execs (tracking deletetlds sean tlds/faces/)
		Quit( True )
	}
}
```

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

2-D	2-dimensional
3-D	3-dimensional
ACT-R	Adaptive Control of Thought Rational
AI	artificial intelligence
API	application program interface
ARIA	Advanced Robot Interface for Applications
ART	Adaptive Resonance Theory
CD-ROM	compact disk–read-only memory
COBOL	common business-oriented language
DFT	Decision Field Theory
DOS	disk operating system
FOV	field of view
GUI	graphical user interface
HOG	Histogram of Oriented Gradients
HSV	hue, saturation, and value
ID	identification
I/O	input/output
LBPH	LocalBinaryPatternHistograms
IP	Internet Protocol
LTM	Long-Term Memory
PCA	Principle Components Analysis
PTZ	pan–tilt–zoon
RGB	red–green–blue
ROI	regions of interest
SS-RICS	Symbolic and Sub-Symbolic Robotic Intelligence Control System

subsim	subsimprocessor
SVM	Support Vector Machine
TBD	to be determined
TCP	Transmission Control Protocol
TLD	tracking/learning/detection
VFW	Video for Windows
VS	Visual Studio
WM	Working Memory
XML	extensible markup language

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO L IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

1 (PDF) DIR USARL RDRL HRF B T KELLEY

INTENTIONALLY LEFT BLANK.