

GUINNESS

Phase I



Marjorie Skubic
Craig Bailey
Sam Blisard
George Chronis

Charlie Huggard
Robert Luke
Grant Scott
Matt Williams

*Computer Engineering and Computer Science Department
University of Missouri-Columbia*

June 20, 2003

Table of Contents

I.	Introduction	(Skubic).....	3
II.	Architectural Overview	(Skubic).....	4
III.	Speech Interface	(Williams).....	8
IV.	Cortex	(Luke).....	11
V.	Spatial Reasoning with SRserver	(Blisard).....	15
VI.	Object Recognition with ImageServer	(Scott).....	19
VII.	PDA Sketch Interface	(Bailey).....	24
VIII.	Spatial Behaviors and “robot_spatial”.....	(Chronis).....	34
IX.	Demo Scenario Results	(Williams & Bailey).....	35
	References		40
	Appendix A: Documentation on how to start up Guinness ..	(Blisard).....	42

ACKNOWLEDGEMENTS

For this work, we received financial and technical support from the Naval Research Lab. The support is gratefully acknowledged, and we especially thank Alan Schultz, Dennis Perzanowski, Bill Adams, and Scott Thomas.

I. INTRODUCTION

In January, 2003, we embarked on the adventure of creating a new mobile robot testbed in which many of our component technologies would be integrated for testing and further development. In particular, we wanted an integrated testbed for studying human-robot interaction issues and extending our spatial reasoning work; thus, we needed a robot system which supported an extensive operator interface. The plan in creating this testbed was to start with an existing robot architecture developed at the Naval Research Lab which already incorporated our spatial reasoning module, and add our new components in vision-based object recognition, sketch understanding and sketch-based navigation. In this report, we will document the robot testbed, describing how the starting architecture was morphed into its existing form and laying out a plan for continued development to support future research.

This effort involved the hard work of everyone listed on the title page, as well as the thoughtful advice of Professor Jim Keller, whose spirit undoubtedly contributed to the selected name of our robot project, “Guinness.”



Figure 1. Guinness and Gang.

From left to right

George Chronis, Grant Scott, Dr. Marjorie Skubic, Matt Williams,
Craig Bailey, Bob Luke, Charlie Huggard and Sam Blisard.

II. ARCHITECTURAL OVERVIEW

In this section, we describe three versions of the robot control architecture: the old, the new, and the planned, as represented in Figures 2, 3, and 4.

The starting point for creating Guinness was the robot architecture developed at the Naval Research Lab (NRL), as shown in Figure 2. The NRL system supports semi-autonomous mobile robot control which includes map building [Adams 2000], planning [Schultz 1999], dynamic obstacle avoidance, spatial reasoning [Skubic 2002b, 2003b], and an extensive operator interface supporting speech, gesture, PDA, and window graphical user interfaces (GUI) [Perzanowski 1999, 2001]. The spatial reasoning module *SRserver* was developed at MU but had already been integrated into the NRL system. The underlying map representation is the evidence grid map in which range sensor readings are accumulated over time to acquire evidence that a grid cell is occupied or not [Martin 1996]. Navigation uses a potential field strategy; the component *trulla* performs the planning of a route from the current position to a new specified position, and *vfh* handles the obstacle avoidance along the planned route.

Several different modalities are supported to provide an operator interface. Commands can be spoken using the speech interface which is a combination of speech recognition and natural language understanding. Speech recognition is accomplished through the commercially available ViaVoice system which converts speech to text strings. The text strings are then converted to robot commands through a natural language processing system developed at NRL called NAUTILUS (Navy AUTomated Intelligent Language Understanding System) [Wauchope 2000]. Although the NRL system also supports gesture, GUI, and PDA interfaces, only the NRL speech interface was used for Guinness, Phase I. Sec. III describes the speech interface port to MU and the changes made to support new capabilities.

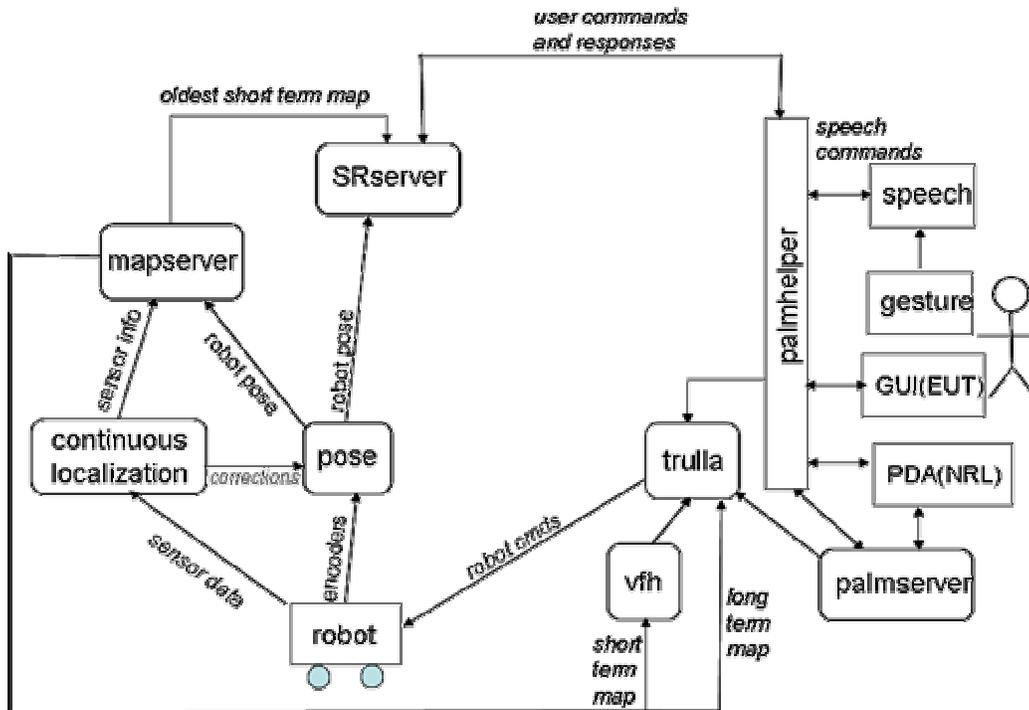


Figure 2. The Starting Architecture, from the Naval Research Laboratory

Figure 3 represents the new Guinness architecture at the end of Phase I. The evidence grid map representation has been retained, and the system still supports the same planning, map building, obstacle avoidance, and spatial reasoning. Additions to the *SRserver* module are described in Sec. V. A major addition to Guinness is the *imageserver* which provides capabilities for object recognition using computer vision (see Sec. VI). Also, the MU PDA sketch understanding interface is used instead of the NRL PDA interface. Sketch information from the PDA is sent to a sketch server and processed via the *spatialview* program which generates log files (see Sec. VII). The log files are then read by the *robot_spatial* program which performs the sketch-based navigation control to guide the robot along the sketched route (see Sec. VIII). To support these new interfaces, the old *palmhelper* module has been morphed into a new interface module we call the *cortex* (see Sec. IV for a list of new commands). Note that the gesture, the EUT (a GUI), and the NRL PDA interface support have not been eliminated from Guinness Phase I; however, these interfaces have not been tested at MU.

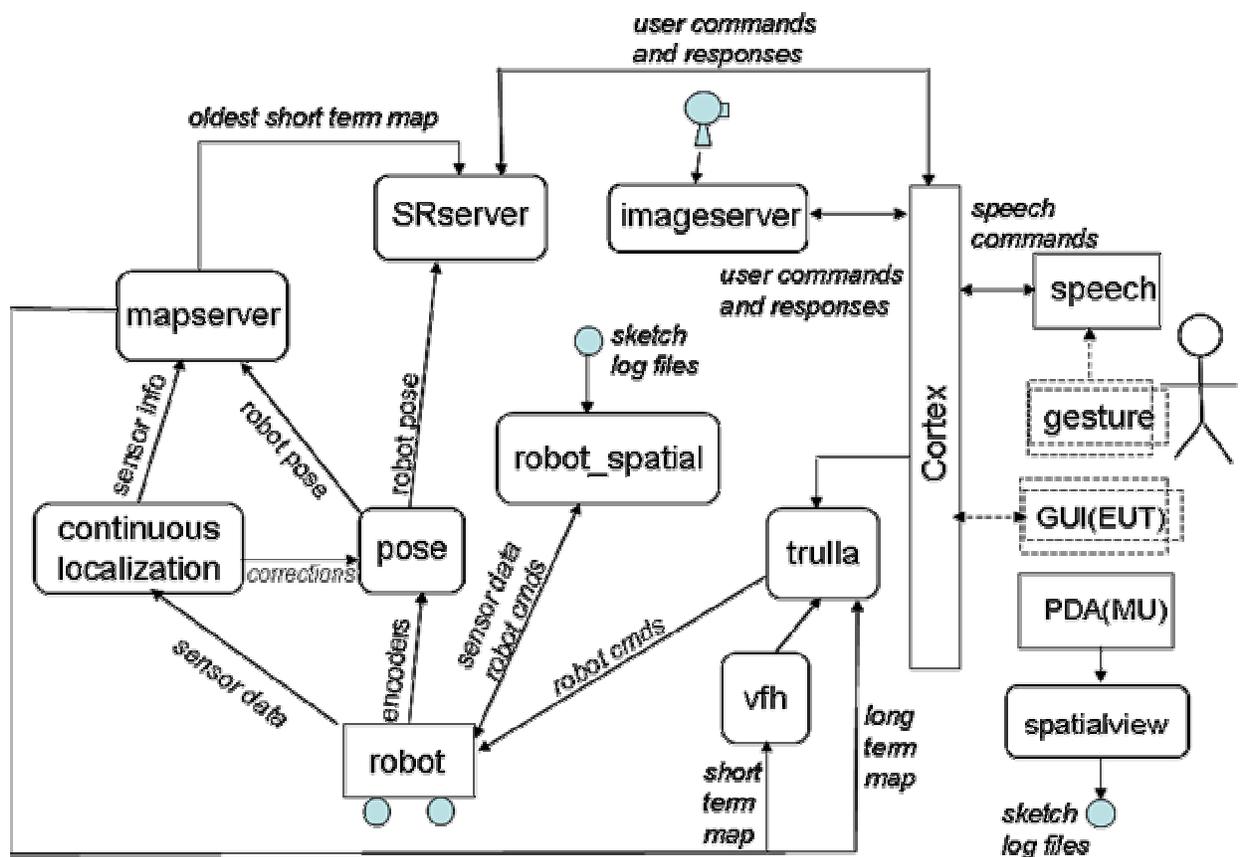


Figure 3. The Current Phase I Architecture

Figure 4 illustrates the additional changes that are planned for Guinness which are primarily adding more connections between modules for further integration. For example, there will be a link between *SRserver* and *imageserver* to support the automatic labeling of segmented objects using computer vision object recognition. In addition, the sketch interface will be integrated through the *cortex*, eliminating the intermediate log files. The route information will be extracted from the sketch and sent to the *cortex* for transmission to the robot for route navigation. The *robot_spatial* module will be replaced with a server called *spatial_behaviors* which will execute the sketched route and also support navigation commands in the form of “move-while” or “move-until” a specified spatial condition has been reached. The existing map building, planning, and obstacle avoidance via *vfh* will be retained. However, to provide a common command link to the robot, a low-level obstacle avoidance module will be integrated as well. The planned integration is intended to retain support of the existing NRL and MU functionalities while at the same time incorporate additional capabilities as outlined above.

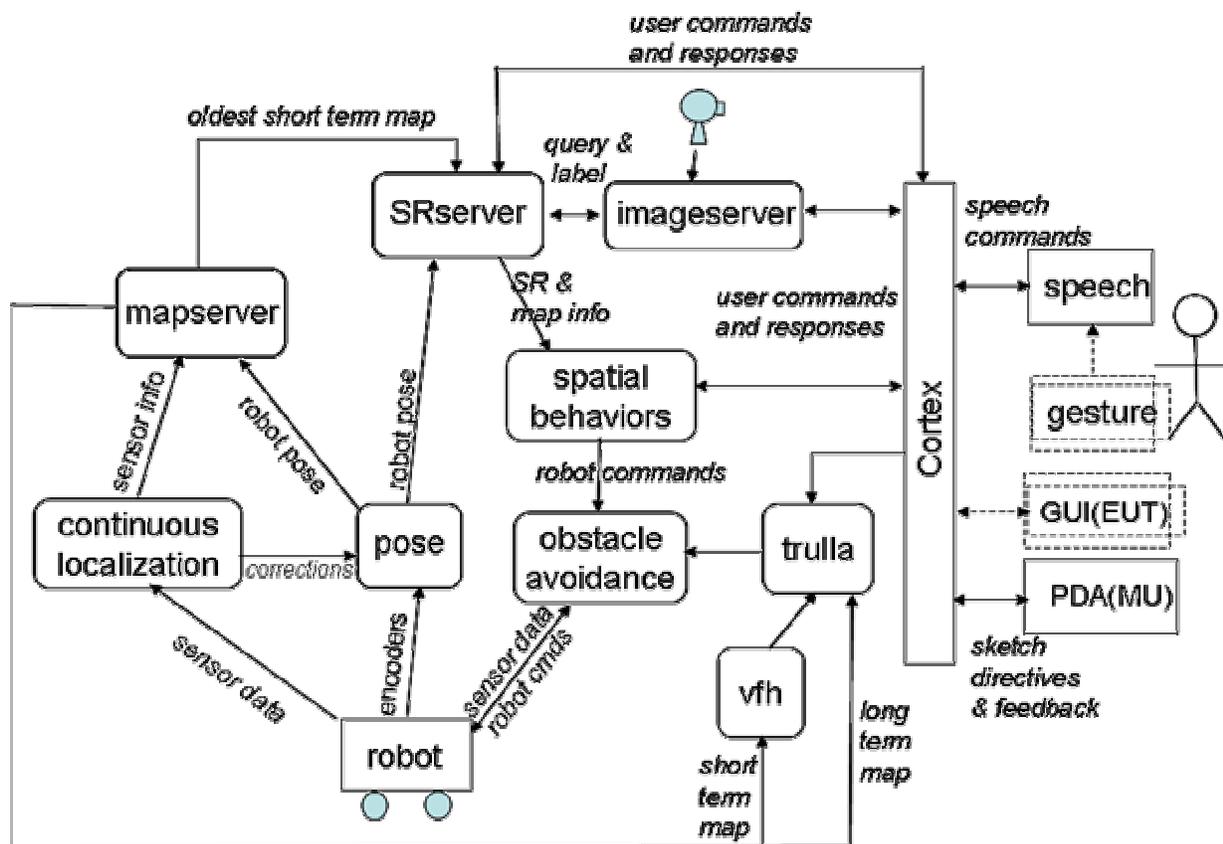


Figure 4. The Planned Architecture for Future Research

Figure 5 shows the hardware components and the mapping of software module to hardware component. The Nomad has an onboard computer with a Pentium processor. The low level motor controls run on this system. An Ethernet switch attaches the Nomad's computer to a laptop sitting on top of the robot. This laptop runs most of the programs used in the Guinness system, as shown in Figure 5. Two desktop computers are used to disperse the processor load. A Windows-based computer controls all speech interface software including Via Voice, Reco and Nautilus. A second desktop running linux is used for SRServer and displays map output for the system user. Wireless networking is used to connect the laptop software to that on the desktops.



Figure 5. Hardware components used for Guinness.

III. SPEECH INTERFACE

There are three major parts to the Guinness speech interface.

- CLISP code/executable (with foreign functions)
- Viavoice/RECO (sends viavoice messages to NAUTILUS)
- NAUTILUS code from NRL (processes text into robot commands)

CLISP

NRL uses NAUTILUS under Allegro - which is a commercial common lisp implementation. We decided to use a free version of common lisp called CLISP, which is available on the internet. It must be customized to use the foreign functions (FF) from NAUTILUS. Allegro can directly access Dynamic Linked Library files (.dlls) - CLISP cannot. The only way to incorporate these foreign functions written in C is to compile the CLISP Lisp executable along with the foreign function code. In a way, this is similar to making them new Lisp functions within the Lisp environment, although they still need to be loaded as foreign functions. Unfortunately, this was a difficult task. It could be done dynamically on Unix, but not Windows. For this reason, not many people use foreign functions in CLISP on Windows.

The first step in this process was downloading and compiling the original CLISP source code. Unfortunately, the code we downloaded did not compile easily - we had to make many changes to finally get it to work correctly. The common problems involved little things such as replacing 'true' with TRUE, 'false' with FALSE, and similar such things. These errors were most likely a result of compiling in the Microsoft Visual C++ environment.

The next step was to incorporate the foreign functions. These functions allowed CLISP to execute functions written in C. This code was given to us from NRL and was used by NAUTILUS to communicate with other machines (SRServer, Imageserver, and NServer). The process involved compiling the FF Lisp definitions within the Lisp environment into a C source code file. This file is then compiled with all of the Lisp code and the actual foreign function code, which results in a new Lisp executable. This Lisp executable can then access the foreign functions after the definition file is loaded.

Viavoice / RECO

ViaVoice is a commercially available speech recognition program. We used the ViaVoice SDK to build and compile our own grammar which allowed us to greatly restrict the sentences and commands that ViaVoice would try to recognize. With a much smaller grammar, the recognition accuracy of the program increased significantly.

RECO is the program which receives messages from ViaVoice and sends them to NAUTILUS when requested. This program is originally from IBM and an example is included in the ViaVoice SDK. We used a version which had been modified by NRL to work with NAUTILUS. NAUTILUS sets itself up as the server and then RECO accesses it as a client. When NAUTILUS is expecting speech input, it constantly checks RECO for any new commands.

The only change to RECO was that CLISP NAUTILUS could not handle a null string being sent to it. The solution we used was to send an exclamation point (“!”) back to NAUTILUS instead of a null string. Then, NAUTILUS was altered to ignore such strings.

NAUTILUS

NAUTILUS is the natural language processing element of the interface. It takes a text string and converts it into a logical form which can be evaluated directly in Lisp into commands to send to the robot. This section will cover three things – the port from Allegro to CLISP, the major changes, and then additions.

A. Porting Nautilus to CLISP

There were numerous issues involved in porting to CLISP. Without going into too much detail, the common problems involved:

- 1) Load functions – Some of the Load functions in the original code did not work well in CLISP. We replaced them with load functions that work in CLISP.
- 2) Pathnames – Pathnames seem to be treated differently in Allegro and CLISP. Some code specifying file locations had to be rewritten. Also, the original code could automatically add “.lisp” to the end of file names. This would not work in CLISP and the “.lisp” had to be added.
- 3) Confirm-modules – Many files call a ‘confirm-modules’ function near the beginning to confirm certain relevant modules have been loaded. Due to the above two problems, this code did not work in CLISP and could not be fixed. The solution was to explicitly load all modules at the beginning of compilation.
- 4) (EXCL::BQ-COMMA FOO) was used several times in the code and was not recognized by CLISP. The solution was to use “,NAME () ”.
- 5) All foreign functions required a prefix of “common-lisp-user::”.
- 6) The ‘get environment variable’ function needed to be system::getenv
- 7) CLISP handles strings indexes differently than ALLEGO. CLISP returns an error when the program tries to access an index outside of the string (including 0 for a null string). Apparently ALLGERO did not. This caused a few errors in NAUTILUS.

B. Major changes

The biggest problem we had to address was how to handle humans within the NAUTILUS system. There are three relevant categories in the grammar – objects (table, pillar, etc), systems (Guinness, Magneto, etc), and r-observer (Dennis, Matt, etc). As part of this project, we wanted to be able to label and interact with humans. Ideally, this would have been done with r-observers. However, it would have involved many changes to the NAUTILUS grammar, which we did not have the ability to change at the time.

Our initial solution was to label humans as systems. The grammar already supported this (“We could say ‘Guinness face Bob’”). This was not grammatically correct if Bob was an object and therefore would not be processed by NAUTILUS. As an object, the sentence would need to be “Guinness face the Bob”). Unfortunately, this led to other problems within NAUTILUS as the existing functions are mostly made to interact with objects. Incorporating systems led to many additions and special cases within the code. In the end this did not turn out to not be a very robust answer to the problem.

The final solution was to label humans as objects – which meant that they would work with the original code. To solve the grammar problem, we parse an input for a human name (“Bob”) and add an article before it (“a Bob”). Any speech output from the robot is also filtered to produce grammatically correct sentences. The only problem with this solution is that all human names must be hard-coded in the parsing/filtering functions.

Another issue is that NRL’s system is closely tied to gestures. Since we do not have the ability to detect gestures, we removed or ignored most of this code. This caused a problem with some functions, such as the ‘explore’ command – which always needs a gesture. This problem was solved by adding an ‘explore here’ command which tells the robot to explore in an area predefined by the user.

C. Additions

We added three functions to the existing NAUTILUS system.

- Remove a label
- Rename a label
- Do you recognize anything
- Do you recognize Bob

‘Remove a label’ contacts srserver and attempts to remove the given label name from the map.

‘Rename a label to newlabel’ contacts srserver and attempts to rename the given old label to a new label.

‘Do you recognize anything’ queries ImageServer and checks all available networks for confirmation of the specific target.

‘Do you recognize Bob’ queries ImageServer and checks the current frame against the specified target.

IV. CORTEX

Cortex is a virtual router that relays commands between modules of the robotic system WAX. Some commands are simply relayed from a sender to a receiver by the Cortex. Other commands are acted upon by the Cortex itself, such as moving the robot forward a given distance.

Cortex works over TCP/IP connections to several other modules. These modules consist of SR Server, Speech, palm, Image Server and Trulla. Cortex acts as a server to which the Speech module connects, but it acts as a client to every other module. This leads to an order of startup for the modules. This does not however lead to a great instability of the whole system. If a module does crash, the module can be restarted and Cortex will most likely be able to reconnect without reboot.

Cortex is able to relay or act upon many different kinds of commands such as palm sketches, body gestures and GPS information; but the bulk of the commands come from speech. There are around forty different commands that can come from speech ranging from zeroing the robot, moving to a location, or even to recognizing objects using the onboard camera. The full list of command tokens and their actions has been added to the end of this section.

There was only one change to Cortex that was fully implemented this semester. This consisted of connecting Image Server to Cortex and allowing lanes of communication between it and the other modules. The user is able to ask if the robot recognizes anything in front of it. This command is passed to Image Server for processing and the returned information is spoken by the robot. This was a simplistic first step in the development of more complex future commands for Image Server.

Along with the completion of the first commands to Image Server, ground work for future lines of communication with Spatial Behaviors and a Working Memory have been developed. Spatial behaviors will consist of movement commands based on qualitative states of the robot's proximity. When the Spatial Behaviors module is ready to be placed in the system, Trulla will be decoupled from the robot's actual movement. The output of both Trulla and Spatial Behaviors will go to a collision avoidance module so that the two navigation modules will not conflict.

Because some of the desired future actions of the robot are repetitive, it is necessary to create a module for the robot to remember its current action. This is where the working memory, which was alluded to above, comes into play. It is a state machine that holds the current action of the robot and commands the proper module to perform that action. The Working Memory then listens for a return from the module and changes its state appropriately with that return.

Cortex Commands

Command Protocol	Action/Result
connect_robot <Nserver machine name> <Nserver port> <robot name>	Causes this process to contact the Nserver
real_robot	Causes the Nserver to switch from simulated robot to the real robot
initialize_robot	Initializes the robot's sensors and sensor mask.
token <token_number> <integer_arg> <string_arg>	Make the robot perform some particular motion or task. See table below for a list of tokens.
gs	Gets the robot's pose and returns it (tenths of inches, tenths of degrees)
tk <message>	Makes the robot speak the <message>.
palmpointrequest	Gets the most recent gesture from the palmpilot (if one is available).
gesturepose <x> <y> <s> <t> <gtype> <gvalue> <gerror> <gtime>	Informs palmhelper that speech considered the provided gesture when the robot was at the given pose.

Cortex Command Tokens from speech

token (defined value)	Action/Result
Zero (1)	Zero the robot
GoToObject (3)	Go to a specific segmented object
GoGesture (4)	Go to a gestured location
FollowMe (5)	Follow the user
FaceObject (7)	Face a specified (x, y) location
FaceMe (8)	Face the user
MoveToMe (9)	Move to the user
TurnToObject (10)	Turn to a specified object
LabelCoord (11)	Label a segmented object in the scene
MoveForward (13)	Move forward a specified distance
Turn (15)	Turn a specified degree
GoWaypoint (17)	Move to a specified object
GoToCoord (18)	Move to a specified (x,y) location
Quit (19)	Stops the current process
GoThroughDoor* (20)	Move through a specified doorway
GoLocation (21)	Move to a specified (x,y) location but makes sure (x,y) is a valid location
ExploreAreaVector (22)	Explores along a given vector
GoToPalmXY (23)	Move to a specified (x,y) location but makes sure (x,y) is a valid location
FollowPathPalm (24)	Follow a path given by palm

ExploreAreaPoints (25)	Explores a given area
ShowMap (26)	Sends a map to palmServer
ExplorePerimVector (27)	Explore around perimeter
ExplorePerimPoints (28)	Explores the area
ChemSensorOn (29)	Sets ChemSensorOn = 1 which allows a polling of chem sensors
ChemSensorOff (30)	Sets ChemSensorOn = 0
GoBack (31)	Move to previous location
GPSExploreAreaVector (32)	Explores along a given vector given by GPS
GPSGoToPalmXY (33)	Move to a specified (x,y) location given by GPS but makes sure (x,y) is a valid location
GPSFollowPathPalm (34)	Move along a path given by GPS
GPSExploreAreaPoints (35)	Explores an area given by GPS
GPSExplorePerimVector (37)	Explore perimeter given by GPS
GPSExplorePerimPoints (38)	Explore perimeter given by GPS
WeatherQuery (39)	Post weather to palmServer
FindObject* (40)	Go to each object in the area and label them using ImageServer and SRServer
DoYouSee* (41)	Rotates and labels objects using ImageServer and SRServer
Recognize (42)	Recognize a specified object or any object in front of the robot.

* Commands are not currently implemented

V. SPATIAL REASONING WITH SRSERVER

SRserver is the core of the spatial reasoning system developed by the University of Missouri-Columbia and the Naval Center for Applied Research in Artificial Intelligence at the Naval Research Laboratory in Washington, D.C. This piece of software draws heavily on work previously done by Dr. Pascal Matsakis et al involving the Histogram of Forces [Matsakis 1999, 2001] [Skubic 2003a].

SRserver communicates to the rest of the world through a TCP/IP connection on `PORT_SPATIAL`, in the parlance of the Wax system. It is a standalone server and generally only communicates with Nautilus and the Mapserver processes. It retrieves from the mapserver evidence grid maps which are subsequently processed through the use of a blur mask and a segmentation algorithm [Skubic 2003b]. After this process is completed, we are ready to use the histogram of forces to do some interesting things with our evidence grid maps.

Initially, SRserver was used to generate linguistic descriptions about the objects in the evidence grid map and their relationship to the robot. For example, if you ask SRserver what it sees, it will come back with a natural language response of “I see objects to the left of me and to the front.” This initial capability is quite powerful and is a base for the rest of our work. In order to compute these linguistic descriptions, we use a fuzzy rule base in conjunction with Matsakis’ histogram of forces [Skubic 2001b].

In addition to enriching our robot dialog with linguistic descriptions of the environment around the robot, we also have the capability to utilize the histogram of forces to generate points around the objects that most would agree are to the left, right, front, or behind the object in question from the point of view of the robot. In order to create this interaction, we extract the main direction from the histogram of forces and use it to find our front and rear points for the object in question. We then look at the perpendicular of the line going through the midpoint between the front and behind points in order to generate the left and right points in a similar fashion [Skubic 2002c].

These capabilities would be useless unless we’re able to utilize the information SRserver generates in a meaningful fashion. As previously mentioned, this is accomplished by using TCP/IP communications on port 5021 (this is Wax’s `SPATIAL_PORT` and can be changed). In order to get SRserver to respond through this communication mode, you must setup communications with it by using the Wax Comm package (namely `robust_read` and `robust_write`). For examples on how to set up these links for communication, look at the democlient source code in the appropriate SR directory.

The following table lists the possible commands that one can issue to SRserver from a remote client and the action that will occur:

<i>SRserver Command Tokens</i> Command Token	Action/Result
GET_NUM_OBJECTS	Segments and labels the current grid map and returns the number of objects
GET_HL_DESCRIPTION	Returns the high-level description of the map from the robot's perspective
GET_NEAREST_OBJECT [DIR]	Returns the nearest object in the direction DIR
GET_DET_DESCRIP [OBJ_NUM POBJ_NAME]	Gets a detailed description for object number OBJ_NUM or persistent object named POBJ_NAME
ASSIGN_LABEL [OBJ_NUM] [LABEL]	Assigns object number OBJ_NUM the label LABEL
ASSIGN_LABEL_DIR [DIR] [LABEL]	Assigns the label LABEL to the object in direction DIR
GET_OBJ_DESC [OBJ_NUM POBJ_NAME]	Gets the linguistic description of object number OBJ_NUM or the persistent object labelled POBJ_NAME
GET_RIGHT_POINT [OBJ_NUM POBJ_NAME]	Finds a point to the right of object OBJ_NUM or persistent object POBJ_NAME
GET_LEFT_POINT [OBJ_NUM POBJ_NAME]	Finds a point to the left of object OBJ_NUM or persistent object POBJ_NAME
GET_BACK_POINT [OBJ_NUM POBJ_NAME]	Finds a point in the back of object OBJ_NUM or persistent object POBJ_NAME
GET_FRONT_POINT [OBJ_NUM POBJ_NAME]	Finds a point in front of object OBJ_NUM or persistent object POBJ_NAME
GET_LRFB_POINTS [OBJ_NUM POBJ_NAME]	Finds all four points around an object OBJ_NUM or persistent object POBJ_NAME
*GET_BETWEEN [OBJ_1] [OBJ_2]	Finds a point between OBJ_1 and OBJ_2

* Command is in SRserver v1.5.5	
OBJECT_RENAME [ORIG_LABEL][NEW_LABEL]	Renames persistent object ORIG_LABEL with NEW_LABEL
OBJECT_DELETE [LABEL]	Deletes persistent object named LABEL

Error/Signal Handling in SRserver is handled the same way as it is in all of Wax. On a ^C, the server exits gracefully and closes all network connections that it currently has open. Note that you need to make sure to kill off ALL processes in the order that they were started; otherwise you will encounter hanging sockets.

Also, SRserver has checks for garbage input of server commands. Beware that there is a certain sequence of commands that must be entered in the command string in order to communicate effectively, and that the lack of certain key words in the command string can lead to a "garbage" command. See democlient.c for examples on how to effectively communicate with SRserver.

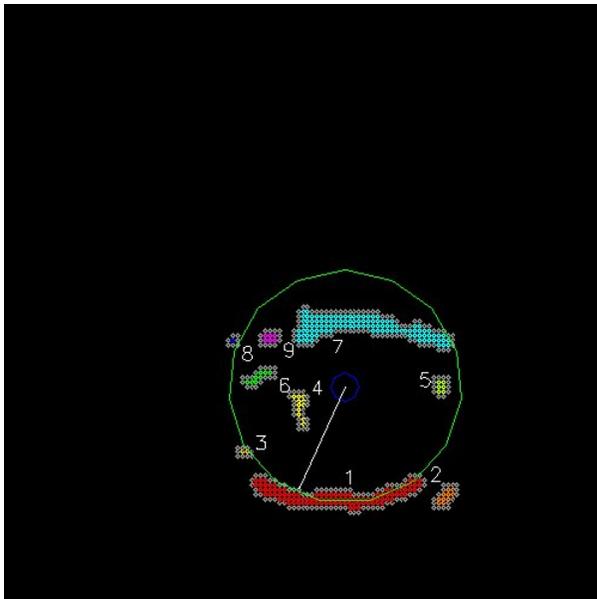


Figure 6.a
Segmentation of objects in
the scene.

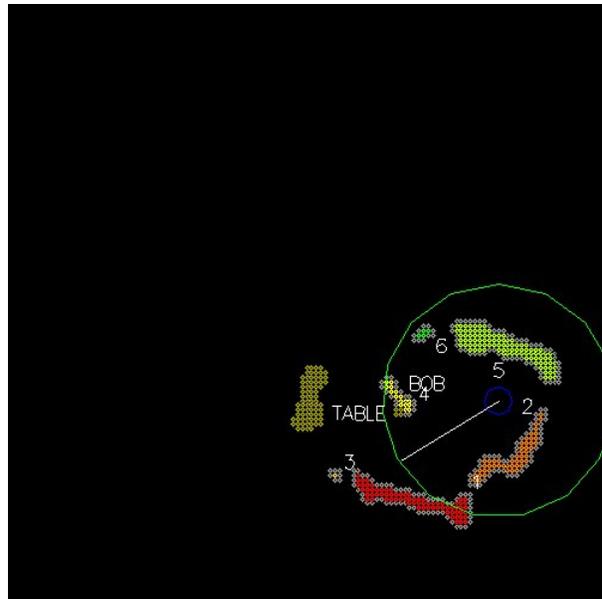


Figure 6.b
Two segmented objects have
been labeled.

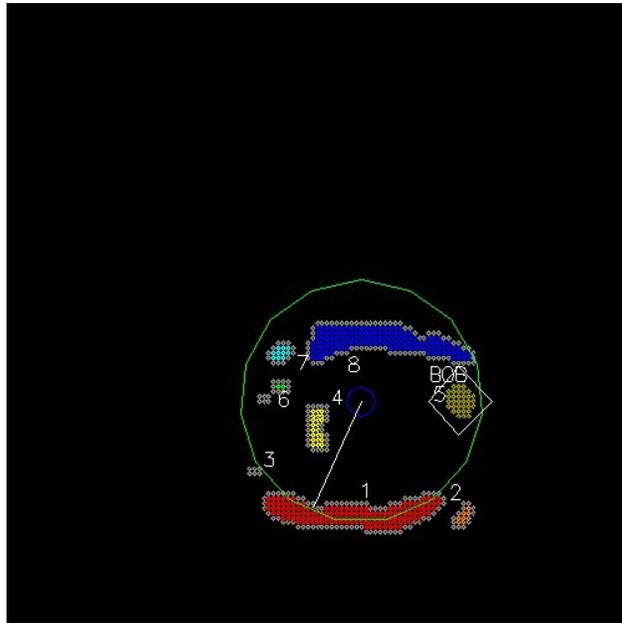


Figure 6.c
Area around Bob used for spatial
“go to” commands is highlighted.

VI. OBJECT RECOGNITION WITH IMAGE SERVER

ImageServer is a robot vision module capable of capturing and processing image scene data. The core image processing ability of ImageServer is designed around the Morphological Shared-Weight Neural Network [Won 1995,1997]. Image server responds to processing requests with verbal phrases stating what target and the angle from center where it was detected. ImageServer is designed as a robust and fault-tolerant TCP/IP server, providing information upon demand from other modules. ImageServer implements a wide range of error and signal handling abilities to allow it to be resilient to dropped connections, connection timeouts, and many other issues that may arise during execution. To control and modify ImageServer during run-time, a separate Controller entity was written. The controller can modify, through the query channel, default settings inside ImageServer while the program is running. ImageServer also has the ability to either read files from disk, or capture image from a range of USB web cameras. Two primary files are needed to configure ImageServer upon startup, the database file and the parameter definition file. The remaining parts of this section elaborate on the various components of ImageServer and examples of the database file and parameter definition file are included. The communication protocol used to issue commands to ImageServer concludes this section.



Fig. 7.a

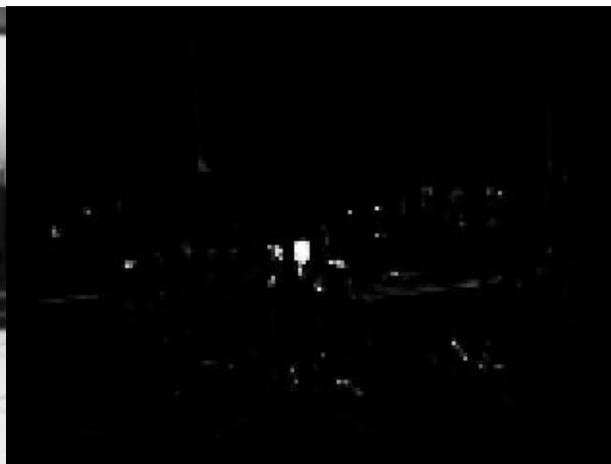


Fig. 7.b

As stated previously ImageServer uses a Morphological Shared-Weight Neural Network (*MSNN*) as the primary vision processing mechanism; this is followed by some processing of the MSNN output. The MSNN is a two-stage heterogeneous neural network model, similar to the architecture known as convolutional networks. Previous research has shown the MSNN to be a robust recognition technology, capable of performing well despite low-resolution images and under various target orientations and occlusions [Scott 2003]. The first phase of the MSNN performs a feature extraction and under sampling on the input sub-image. The feature extraction phase produces the direct input of the classification phase. The classification phase is implemented as a fully connected to feed-forward network. The output of the classification phase is used to construct a field of confidence values over the input image space, where each pixel position represents the confidence of target object center. Figure 7.b is the confidence field from a scan of Figure 7.a. The confidence field is converted to a binary image through thresholding, which is further processed. The final processing consists of a connected

component algorithm, accompanied with a determination of the strongest signal, or largest blob size. The largest blob center over a size threshold is inferred to be the location of the object upon which the current scanning MSNN was looking for. Figure 7.c shows the output image produced by superimposing the thresholded binary image over the input image. The MSNN in this instance was trained to recognize the stuffed animal “wiley,” Figure 7.c also shows a typical MSNN false alarm on the arm under the right arm. Figure 7.d is an output image that draws a box the size of the network input around the coordinate of the center of the largest connected component, due to post processing the false alarm observed in Figure 7.c has no effect on the end response of ImageServer. ImageServer responds with the angle of the detection, in the case of this processed image, the coordinates were (153,124). The angle is calculated by the following equation:

$$\text{angle} = (\text{IMG_X_DIM}/2 - \text{xPosition}) / \text{IMG_X_DIM}/2 * (\text{VIEWABLE_ANGLE}/2).$$

Based on the equation above, ImageServer determines the angle to be 0.9625 degrees, and returns an angle of 1 degree. ImageServer and Cortex have an understanding of straight ahead being 0 degrees, and left being positive angles.

To handle errors that may occur, particularly communication errors with the clients, a wide range of signal handling capability is implemented in ImageServer. When each incoming client connection is passed to a processing thread, that thread inherits the entire signal handling ability. This prevents child signals from interfering with the processing of the core ImageServer process and any other sibling threads that may be executed. Additionally, the connection handling and thread spawning for the two channels are separated into their own respective threads. To facilitate communication and allow child processes to mutate the run-time parameters of ImageServer, many of these variables are stored in shared memory segments created and initialized by the core parent.



Fig. 7.c

Fig. 7.d

Two communication channels are implemented in ImageServer, a query and persistent channel. Both channels are implemented in a threaded fashion to allow multiple concurrent client connections. The query channel provides access to ImageServer’s full command set using a single command per connection socket. The persistent channel is primarily for the Cortex

module and is solely for the image processing commands. The persistent channel implements a keep-alive connection with the client that does not timeout. In contrast, the query channel enforces a 30 second timeout after connection is established to receive the client command. The ImageServer controller is a TCP/IP client that connects through the query channel. The controller has the ability to update the confidence threshold, the confidence blob size, and force the ImageServer to restart completely or shutdown all connections and exit.

To acquire images to process, ImageServer has two primary methods: using a disk file or frame grabbing from a USB camera. When ImageServer is started in file reading view, a specific file is constantly over written with a new image by some outside source. ImageServer continually processes the same filename upon each client request; the required file type is Portable Gray Map (PGM). When started with video capture enabled, ImageServer acquires the image from a frame grab buffer and converts it into a gray-scale single byte per pixel image. ImageServer creates output images modes of operation; during image capture the captured image is also written out. The processed images that are produced by ImageServer are the binary threshold image superimposed over the raw input image and a detection image plane of the confidence values. An additional capability of ImageServer is the ability to display one of the two processed images using a suitable viewer. All output images from ImageServer are in the PGM format.

ImageServer is configured at run-time using two primary files: the database file of trained MSNN and the parameter definition file. The database files parsed to define the number of objects known to ImageServer, and their respective MSNN scanning and weight files. The scanning file defines the structure of the MSNN file used for the object, and the weight files provide all the trained connection weights to re-initialize the MSNN. The default database file *IS.db* provides comments with information on proper format. The parameter file is parsed to provide ImageServer various information pertaining to its behavior. The current parameters that can be defined in the parameter file are: *ImgSource*, *CaptDir*, and *ImageViewer*. Examples of the parameter definition file lines that set each of the previously mentioned parameters can be found as comment statements in the *param.conf* file that accompanies ImageServer.

```

                                     IS.db
###
###  ImageServer
###      Database Definition File
### FORMAT> TargetName:ScanFile:WeightFile:
###
bob:scanbob882.net:testbob882@125.wts:
wiley:scanwiley.net:wiley4040.wts:
```

The following is a complete log file from ImageServer during a short test run. Each line has both a [timestamp] and a <process ID> stamp. Most of the log shows the messages from ImageServer as it is loaded and initialized, including the parsing of the database and parameter files. This is followed by an instance of the “IS_WHAT” command where it detected Bob and Wiley, and sent the output images to the screen using the image viewer program specified in the parameter file. The log is concluded with the intercept of an interrupt signal and the shut down of ImageServer.

```

[2003-05-14|13:43:46] <1541> Local Host: localhost.localdomain
[2003-05-14|13:43:46] <1541> ImageServer 0.2.4, running on ports: 2003, 5150
[2003-05-14|13:43:46] <1541> Database File: IS.db
[2003-05-14|13:43:46] <1541> Image Display Enabled
[2003-05-14|13:43:46] <1541> Reading database file [IS.db] line:
    wiley:scanwiley.net:wiley4040.wts
[2003-05-14|13:43:46] <1541> Reading database file [IS.db] line:
    bob:scanbob882.net:testbob882@125.wts
[2003-05-14|13:43:46] <1541> Database File: IS.db parsed for 2 targets.
[2003-05-14|13:43:46] <1541> Parameter Definition: ImgSource | Camera </dev/video0> <320>
    <240>
[2003-05-14|13:43:46] <1541> Image Storage Directory set to ImageFiles/
[2003-05-14|13:43:46] <1541> Parameter Definition: ImageViewer | /usr/bin/ee <ee> <SI>
[2003-05-14|13:43:46] <1541> Parameter Definition File: param.conf parsed for 3 parameters.
[2003-05-14|13:43:46] <1541> Parameter Definition File: param.conf
[2003-05-14|13:43:46] <1541> Server Socket Created
[2003-05-14|13:43:46] <1541> Server Socket Bound and Listening...port 2003
[2003-05-14|13:43:46] <1541> Signal Processing Handlers Installed...
[2003-05-14|13:43:46] <1541> Awaiting Client Connections
[2003-05-14|13:43:46] <1541> Persistent Connection Server Socket Created
[2003-05-14|13:43:46] <1541> Persistent Connection Server Socket Bound and
    Listening...5150
[2003-05-14|13:43:46] <1541> Awaiting Persistent Client Connections
[2003-05-14|13:43:52] <1541> Persistent Client Connection Received
[2003-05-14|13:43:52] <1541> Awaiting Persistent Client Connections
[2003-05-14|13:43:52] <1546> Processing Client Commands, on Persistent Channel
[2003-05-14|13:53:02] <1546> IS_WHAT command received
[2003-05-14|13:53:12] <1546> Send scanned image to screen
[2003-05-14|13:53:18] <1546> I Found, wiley at -10 degrees
[2003-05-14|13:53:18] <1546> Send scanned image to screen
[2003-05-14|14:01:18] <1546> IS_WHAT command received
[2003-05-14|14:01:28] <1546> Send scanned image to screen
[2003-05-14|14:01:35] <1546> I Found, bob at -12 degrees
[2003-05-14|14:01:35] <1546> Send scanned image to screen
[2003-05-14|14:02:14] <1546> IS_WHAT command received
[2003-05-14|14:02:21] <1546> Send scanned image to screen
[2003-05-14|14:02:29] <1546> Send scanned image to screen
[2003-05-14|14:02:29] <1546> I Found, wiley at -5 degrees, bob at -13 degrees
[2003-05-14|14:44:30] <1546> IS_WHAT command received
[2003-05-14|14:44:30] <1541> Interrupt signal caught, going down gracefully...we hope.
[2003-05-14|14:44:30] <1546> Interrupt signal caught, going down gracefully...we hope.

```

```

                                param.conf
### ImageServer
### Parameter File
### Image Acquisition Source definition
### Parameter Value [source_opt] [x_dim] [y_dim]
ImgSource Camera /dev/video0 320 240
### Captured Image Storage Location
CaptDir ImageFiles/
###
### Image Viewer Data
### Parameter Value(Path) [Executable] [OutputImageType]
ImageViewer /usr/bin/ee ee SI

```

ImageServer Command Line Switches

- -p#####; modifies the query channel port, default is 2003
- -x#####; modifies the persistent channel port, default is 5150
- -l%%%; specifies a log file name, default is “is-TimeDateStamp.log”
- -d%%%; specifies the database filename, default is IS.db
- -c%%%; specifies the parameter definition file; default is param.conf
- -view; specifies for ImageServer to display an output image after each processing command

ImageServer Communication Protocol, defined in “isProtocol.h”

- IS_WHAT; scans current visual scene against all weight files in the database*
- IS_WHO; scans current visual scene against the weight file for the target of interest*
- IS_LOOK; scans a designated vertical slice of the visual scene against all weight files in the database*
- IS_KILLER; terminates the execution of ImageServer, password required.
- RESTART_IS; commands ImageServer to execute a full shutdown and restart, e.g. when the server program has been rebuilt from modified code.
- RE_DB; commands ImageServer to re-initialize from the database file, e.g. new targets are added
- RE_THRESH; resets the target confidence threshold value used in detection processing
- RE_BLOB_SIZE; resets the minimum target detection blob size required for recognition report
- RE_LOG; cycles the log file to start a new one.

VII. PDA SKETCH INTERFACE

The Sketch Interface Overview

This portion of our Spatial Reasoning with Robots project was to implement an interface within which we can intuitively sketch an environment in which the robot will interact. The sketch will provide the robot with a mapping of drawn objects, and a path with which to navigate those objects.

The challenge of the sketch interface was to intelligently, and consistently, interpret the markings of the user. The goal is to have the user require as little knowledge of the interface as possible. The user should be able to draw and/or mark out objects and paths in such a way that the use of menus or commands is unnecessary. The sketch interface should be able to interpret the markings of the user and transform the sketched image into a map environment within which the robot should navigate.

With work done thus far, a palm pilot based sketch interface that can input a user drawn sketched map has been previously reported [Skubic 2001a, 2002a]. There are some limitations with this early system though. One deficiency requires delimiters to mark each new object. The user must explicitly input these between drawing the objects. The user also had to denote the path in the original implementation. This approach is not an intuitive process for the user. We have redesigned the system such that the sketch interface now determines the different objects and the pathway based upon the drawn sketched markings.

A second limitation was if the user wished to remove an object from the drawn sketch, he had to redraw the entire sketch. It would be more convenient for the user to have the interface interpret a crossed out object and remove it from the sketch, obviating the need to redraw the entire sketch. This operation is now also part of the sketch interface. The user can draw a single crossed 'X' on or near the object that he wishes to delete.

The ability to label the objects in the drawn map was added which allows for further interaction between the user and the robot in the context of its environment. After an object is labeled, the robot can now refer to that object by name. This provides a method to interact with any object recognition systems that the robot may possess. Indeed, we have incorporated such behaviors into our robot system with a morphological shared weight neural network (MSNN) for image recognition.

Several additional user interface features under consideration for inclusion to the PDA interface have been discovered in testing the new user interface with random users in a small usability study. Coloring of the paths and/or objects is of considerable interest as a feedback mechanism to the user. This is easily done on the Palm OS platform, however, our current mix of Palm OS 3.5 on the Handspring Visor and the Palm OS 5 compiler has not allowed this to happen as of yet. The undo feature will also be a valuable addition. This feature will be implemented very shortly. And finally, a way to move an already drawn object would be a convenient feature for the user. This also will be implemented in the short term.

An important component of any user interface into the Guinness system is communications. The preferred platform for our system is 802.11b wireless Ethernet. The PDA communications link has been upgraded to allow this. The sketched map is transmitted seamlessly into an existing TCP-IP based client server system. This allows for real-time sketches of the environment to be input to our robot from a distance. A sketch server program was written which takes the input connection, receives the sketch and writes out a formatted file, similar to

the previous output, but with defined text based object and path delimiters and the object label information.

An important observation that has been made using the new sketched paths within the linguistic description generator, *spatialview*, is the noise introduced by the discreteness of the path pixels. There are many small heading changes that should not be part of the path description, yet due to small variations in the path pixel orientations, we find that the headings generated can introduce small 'jogs' in the final path descriptions. To alleviate this, we will attempt to develop some algorithms to apply curve fitting to the drawn paths. The challenge here will be to determine curved segments from the straight line segments for the smoothing operation. Most likely we will apply some first and second discrete derivatives to determine the amount and acceleration of the heading changes and then apply either a linear line fitting or parametric line fitting as necessary.

The PDA Interface.

The original program that was written to capture user sketches was used as a starting point. The first revision was to create abstract data types for the object and paths as dynamic memory structures. The points of each were then collected as linked lists for easy traversal and storage. The sketched objects themselves were then also placed into a linked list. To ascertain an object from a path, we looked for closed polygons. The distance between the pen down and pen up events was used, and a threshold value set. Currently, if the two are within 18 points in Euclidean distance, then it is considered a closed polygon and marked as such.

The path was then any line drawn such that the ends did not meet each other within the threshold distance. We also wanted to restrict the sketch to only one drawn path. If a second line drawn is determined to also be a path, then the first path is discarded.

To remove an object, we first had to decide what mark would be interpreted as a deletion event. The most obvious choice is to be able to cross out an object or path. The challenge was to recognize the sketched line strokes that signaled the users intentions. The first task was to distinguish a short incoming stroke from a path if it was not a closed polygon, and secondly, to wait for a second short stroke to complete the cross out. This was accomplished by experimentation that most short strokes were composed of fewer than 9 points. This threshold was then applied. Any stroke fewer than 9 points was temporarily held and checked if a second stroke followed.

To determine if two short strokes actually signaled a deletion, and not just random marks on the sketch window, it is needed to determine if they cross. An algorithm based on the intersection of two lines was used. By solving for the following line equations P_a composed of P_1 to P_2 , and P_b composed of P_3 to P_4 , where P_1 to P_4 are the endpoints of the line segments a and b:

$$\begin{aligned} P_a &= P_1 + u_a(P_2 - P_1) \\ P_b &= P_3 + u_b(P_4 - P_3) \end{aligned}$$

it can be shown that:

$$u_a = \frac{(X_4 - X_3)(Y_1 - Y_3) - (Y_4 - Y_3)(X_1 - X_3)}{(Y_4 - Y_3)(X_2 - X_1) - (X_4 - X_3)(Y_2 - Y_1)}$$

$$u_b = \frac{(X_2 - X_1)(Y_1 - Y_3) - (Y_2 - Y_1)(X_1 - X_3)}{(Y_4 - Y_3)(X_2 - X_1) - (X_4 - X_3)(Y_2 - Y_1)}$$

Using u_a and u_b as decision parameters, we can determine if the two line segments cross. If $0 < u_a < 1$ and $0 < u_b < 1$ then the segments cross. If they do cross, it is then just a matter of using u_a or u_b (either one) to solve the equation of the line for the (x, y) coordinate where the segments cross. Note that the denominator of each is the same, reducing the calculations required.

Upon the detection of a crossed line marking the deletion of some drawn object, we must then determine which object the user intended to delete. This is done by searching the sketched objects and path for two criteria. First, find the closest object to the intersection of the crossed lines, and second, does that distance lie within the maximum distance of one of the endpoints to the determined intersection point of the crossed lines. The first object or path that meets this

criteria is then deleted. If none are found, then the crossed marks are removed and nothing else happens.

The last important user interface addition was the ability to label the objects after they are drawn. Initially it was tried such that the labels could be entered and edited directly on the sketched objects in the PDA window. However, there are limitations in Palm OS 3.5 which make this a difficult operation. Indeed, several commercial applications don't make use of this facility. Instead, the objects are drawn with a default number incremented for each drawn object. If the user wishes to change this, they simply tap next to the number, and a dialog box will come up to allow editing of the label. The label can also be re-edited later if so desired. This style of labeling is consistent with that of the commercial applications.

Four menu options are made available to the user. Clear, Transmit, Configure, and Exit. All of these are also available as the Graffiti commands, 'C', 'T', 'F', 'X'. The Configure option is to set the TCP/IP address and port of the receiving sketch server. The Transmit command immediately begins the uplink to the wireless Ethernet, looks for a connection to the sketch server and sends the current sketch information.

Sketch Server

The sketch server is a simple TCP/IP server which waits and listens for an incoming connection for a sketch client. The server currently starts up with only one command line parameter, the port address to bind. The default is 2004. (No particular reason at all.) The server will fork off the connections as they come in and is thus able to handle multiple sessions if ever needed. The command line normally used is the following:

```
sketchServer 3002 > log &
```

This will start the sketch server on port 3002, send all logging information to a file named 'log' and throw the process in the background.

When an incoming connection is received, it waits for a 'skEt' text token to come in to signal that this is a valid sketch client connecting and wanting to send a sketch. It then begins receiving the sketch information. Four other text tokens signify the different types of sketch information, and a termination command.

'OBJC' -- the following information are points of a new object.

'PATH' -- the following information are points of a path.

'LABL' -- the following information is the object labels.

'QUIT' -- the communications session can be terminated, all information sent.

The sketch server assigns each incoming sketch to a file with the filename format: sketch.#.pda, with # a sequential number read in from a text file, sketchFileIndex. If this file doesn't exist, the server will create it and start it at 1. There are no checks made to see if an existing sketch will be overwritten. This shouldn't be the case normally if the sketchFileIndex file is left alone.

Initially, the quality of the transfer of information was less than desirable. There were many dropped connections, as well as dropped data points, especially when using the busy TigerNet wireless infrastructure. It was determined that the TCP sending buffers on the client were probably flooded with too many small packets. To rectify this, the points were 'packaged' into strings of 5 points each. This should increase throughput 500% as well as make less of a demand on the TCP buffer. That is exactly what happened. The transfers are now very efficient and reliable.

ParseSketch

The parseSketch program is a small utility which 'thins' out the points of the sketches for efficient processing by the spatialview program. Originally, the palm interface was connected to a running communications server via serial port, and the points were collected as they were drawn. The speed of the port was slowed purposefully, so as to artificially restrict the number of points input to the receiving program. This isn't possible with the new TCP/IP transfer of information, as the information will be faithfully reproduced on the server side after transmission. The speed restriction also removed points randomly. We felt that it was better to have a controlled system to work within.

parseSketch reads in the file output by sketchServer and examines the distance between each pair of points in the objects and paths. A threshold value is compared and if the distance between points lies under this threshold, then the following point is thrown out, and the next point examined until the distance is greater than the threshold. The threshold values are input via command line parameters such that there is complete flexibility. There are separate threshold values for objects and paths. The command line used to execute this is as follows:

```
parseSketch filename obj_thresh path_thresh
```

The output file will be 'filename.parsed'. Optimal values of the threshold appear to be about 6 for objects and 4 for paths. The values are in pixels and the distances compared are Euclidean distances.

SpatialviewV6

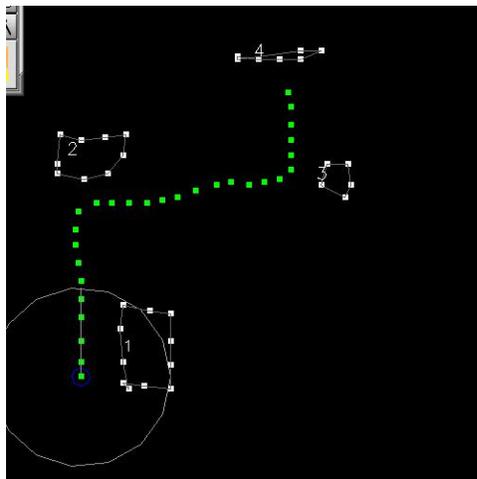
The spatialview program, which inputs a sketched file, and outputs the linguistic path description, was modified and updated to make use of the newer features of the sketch files. First, the ability to input a sketch from the command line into spatialview was added to bypass the current menu system. Second, the new text object delimiters were incorporated into the file parsing. Finally, the label information was parsed from the file. Spatialview was then modified to make use of the new object labels, which are always part of the new sketch file format. The default labeling functions were deleted.

The heading vector was also noted to be incorrect at times when the heading was vertically directed. This was corrected by Dr. Skubic by switching the 'atan' function to 'atan2' and removing the old quadrant checks which proved unreliable.

The SpatialviewV5g was used as the starting point for these modifications. The current plan is to 'merge' the current spatialview program with George's spatialview program that interacts with robot_spatial. The goal is to create a single program that can handle several scenarios. This will also entail the synchronization of the spatial.c files and fuzzy.c files. George Chronis' versions create a modified '.log' file which provides a richer source of information for the robot_spatial algorithms. It is this intermediate '.log' file which will need to be passed along when we implement the sketching system into Guinness.

Additionally, for a graphics class project, I added a 3D component to the spatialview program which allows viewing of the sketched path and objects from the robot egocentric point of view. This provides an interesting and potentially useful interpretation of the sketched environment. This 3D component, as well as the frame grabbing and animation abilities will all be merged into various conditional compiles into a single integrated SpatialviewV7.

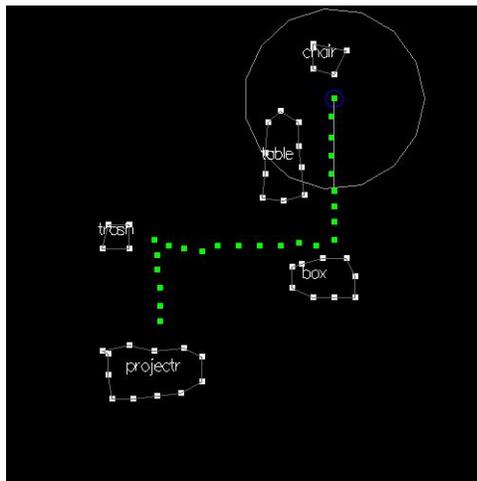
Representative Sketches and Output



START: Move forward
 CHANGE at step 8: Turn right
 CHANGE at step 13: Move forward
 CHANGE at step 21: Turn left
 CHANGE at step 26: Move forward
 END at step 27:
 NUMBER OF PATH NODES = 6

*** PATH DESCRIPTION ***

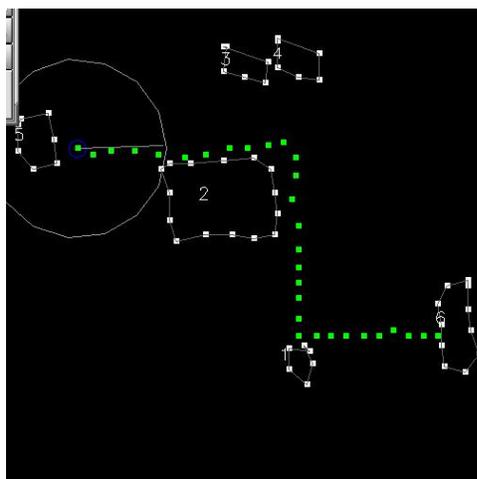
1. When 1 is mostly on the right (and close) Then Move forward
2. When 2 is mostly in front Then Turn right
3. When 2 is to the left rear Then Move forward
4. When 3 is mostly on the right Then Turn left
5. When 4 is in front or mostly in front or in front Then Move forward
6. When 4 is in front (and close) and 3 is mostly to the rear Then Stop



START: Move forward
 CHANGE at step 7: Turn right
 CHANGE at step 12: Move forward
 CHANGE at step 17: Turn left
 CHANGE at step 22: Move forward
 END at step 23:
 NUMBER OF PATH NODES = 6

*** PATH DESCRIPTION ***

1. When table is on the right front (and close) and chair is mostly to the rear (and close) Then Move forward
2. When box is mostly in front or on the left front Then Turn right
3. When table is to the right rear Then Move forward
4. When trash is on the right Then Turn left
5. When projectr is mostly in front Then Move forward
6. When projectr is mostly in front (and close) and trash is mostly to the rear Then Stop



START: Move forward
 CHANGE at step 9: Turn right
 CHANGE at step 14: Move forward
 CHANGE at step 19: Turn left
 CHANGE at step 25: Turn right
 CHANGE at step 26: Move forward
 END at step 29:
 NUMBER OF PATH NODES = 7

*** PATH DESCRIPTION ***

1. When 5 is mostly to the rear (and close) and 2 is mostly in front Then Move forward
2. When 4 is mostly on the left and 3 is to the left rear Then Turn right
3. When 2 is to the right rear Then Move forward
4. When 1 is mostly in front Then Turn left
5. When 6 is in front Then Turn right
6. When 6 is in front Then Move forward
7. When 6 is in front (and very close) Then Stop

Figure 8 These sketches were obtained by random user to gather information for a small scale usability study involving the sketch interface. The sketches are shown as they appear in the spatialview program for path description processing. Example output path descriptions are for each sketched map.

```

sketch.11
Step, X, Y, filtX, filtY, head, obj, pd, dist, cfMD, gfMD, LB, RB
0, 18, 7, 18.0, 7.0
1, 18, 12, 17.7, 17.0, 7.6, 1, 0, 29, 4, 6, 24, 348
0, 18, 7, 18.0, 7.0
1, 18, 12, 17.7, 17.0, 7.6, 4, 11, 7, 262, 260, 304, 206
2, 18, 17, 17.5, 19.5, 3.8, 1, 1, 25, 10, 10, 34, 350
2, 18, 17, 17.5, 19.5, 3.8, 4, 11, 9, 246, 244, 288, 200
3, 17, 22, 17.6, 22.0, 357.7, 1, 1, 20, 16, 18, 44, 352
3, 17, 22, 17.6, 22.0, 357.7, 4, 11, 13, 236, 236, 272, 204
4, 17, 27, 18.6, 26.2, 346.6, 1, 1, 15, 30, 32, 62, 0
4, 17, 27, 18.6, 26.2, 346.6, 4, 11, 17, 236, 236, 266, 210
5, 18, 32, 20.6, 29.4, 328.0, 1, 3, 11, 56, 60, 100, 16
5, 18, 32, 20.6, 29.4, 328.0, 4, 11, 20, 248, 248, 272, 224
6, 23, 33, 24.0, 31.6, 302.9, 1, 5, 12, 96, 96, 134, 58
6, 23, 33, 24.0, 31.6, 302.9, 4, 11, 17, 260, 260, 284, 236
7, 28, 33, 28.4, 32.8, 285.3, 1, 5, 16, 120, 122, 154, 88
7, 28, 33, 28.4, 32.8, 285.3, 4, 12, 16, 268, 268, 290, 238
8, 34, 33, 33.6, 33.0, 272.2, 1, 7, 21, 142, 142, 170, 116
8, 34, 33, 33.6, 33.0, 272.2, 4, 11, 15, 260, 262, 288, 236
9, 39, 33, 38.8, 33.0, 270.0, 1, 7, 25, 148, 150, 174, 126
9, 39, 33, 38.8, 33.0, 270.0, 3, 15, 29, 334, 334, 352, 314
9, 39, 33, 38.8, 33.0, 270.0, 4, 11, 17, 248, 250, 274, 228
10, 44, 33, 44.0, 32.6, 265.6, 1, 7, 30, 158, 158, 180, 138
10, 44, 33, 44.0, 32.6, 265.6, 3, 15, 25, 336, 336, 356, 314
10, 44, 33, 44.0, 32.6, 265.6, 4, 11, 19, 244, 244, 268, 226
11, 49, 33, 49.0, 32.6, 270.0, 3, 15, 21, 326, 326, 348, 300
11, 49, 33, 49.0, 32.6, 270.0, 4, 10, 23, 230, 230, 250, 212
12, 54, 31, 53.8, 33.4, 279.5, 3, 14, 15, 314, 314, 342, 282
12, 54, 31, 53.8, 33.4, 279.5, 4, 9, 25, 212, 212, 232, 194
13, 59, 33, 57.8, 35.2, 294.2, 2, 3, 29, 52, 52, 72, 34
13, 59, 33, 57.8, 35.2, 294.2, 3, 13, 12, 288, 290, 318, 250
13, 59, 33, 57.8, 35.2, 294.2, 4, 9, 30, 194, 194, 210, 178
14, 63, 37, 61.0, 38.0, 311.2, 2, 2, 24, 40, 40, 66, 20
14, 63, 37, 61.0, 38.0, 311.2, 3, 11, 13, 256, 258, 220, 220
15, 64, 42, 62.8, 39.8, 315.0, 2, 1, 19, 36, 36, 68, 12
15, 64, 42, 62.8, 39.8, 315.0, 3, 11, 17, 244, 246, 270, 216
16, 65, 47, 64.0, 42.0, 330.9, 2, 1, 14, 22, 22, 62, 348
16, 65, 47, 64.0, 42.0, 330.9, 3, 10, 22, 222, 224, 244, 200

```

```

START: Turn right
CHANGE at step 11: Turn left
END at step 16:
NUMBER OF PATH NODES = 3

```

```

Step turnrate turncmd
0 0.00 0
1 0.00 0
2 -0.76 -4
3 -1.20 -4
4 -2.22 -4
5 -3.65 -4
6 -4.92 -4
7 -3.53 -4
8 -2.18 -4
9 -0.44 -4
10 -0.88 -4
11 0.88 4
12 1.76 4
13 2.74 4
14 3.00 4
15 0.75 4
16 3.13 4
16, 65, 47, 64.0, 42.0, 330.9, 2, 1, 14, 22, 22, 62, 348
16, 65, 47, 64.0, 42.0, 330.9, 3, 10, 22, 222, 224, 244, 200

```

Figure 9. The output of the spatialview program is saved in a log file. A typical log file is shown here. This file contains the parameters needed to generate the linguistic descriptions and is input into robot_spatial to actually control the robot actions.

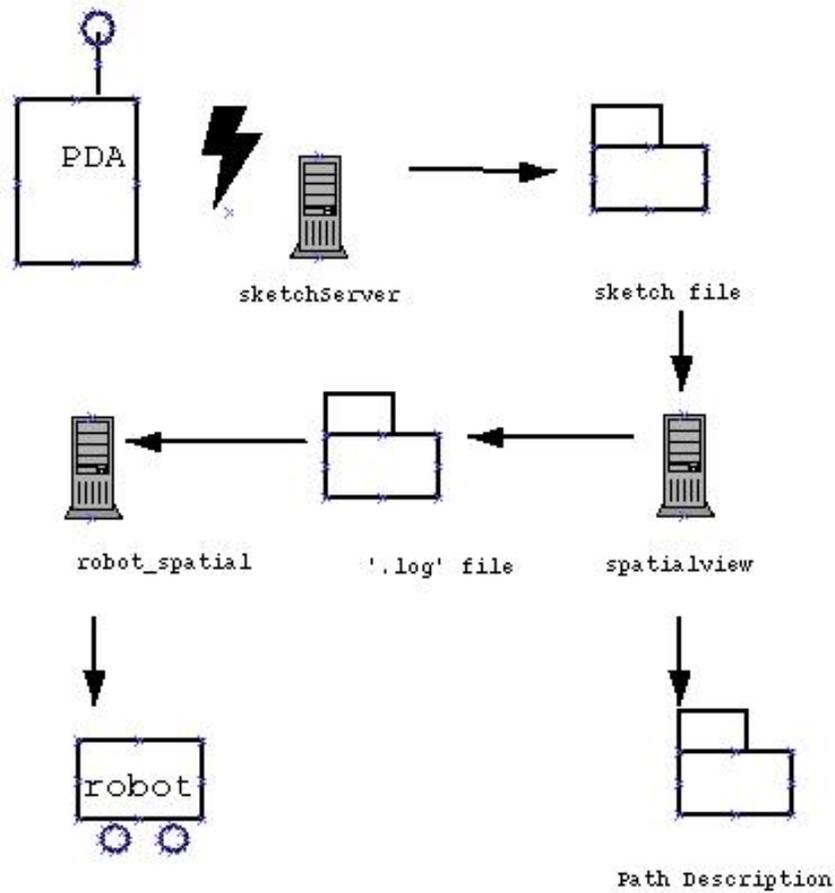


Figure 10. A diagram showing the current architecture of the PDA sketch-based navigation for the Guinness system. Many of these components will be merged into a single server system comprising the sketchServer, a modified spatialview, and the robot_spatial programs. This will pipeline the sketch information quickly and efficiently into robot commands. In addition, the integrated sketch system will connect the spatialview sketch interpretations to the robot_spatial module through the reworked 'Cortex' component. This architecture will be in keeping with the existing framework with the NRL components.

VIII. SPATIAL BEHAVIORS

The spatial behaviors module of Guinness will eventually receive user commands and responses in the form of sketches on a PDA and speech from different Guinness modules. It will also receive evidence grid maps and spatial relations information from yet another Guinness module and sensory information from the robot. The spatial behaviors module is tightly integrated with a low level obstacle avoidance module and is responsible for issuing direct navigation commands to the robot.

The current implementation of the spatial behaviors module will receive a sketch from a PDA and process it, but it does not yet communicate with any other Guinness module. Processing the sketch involves extracting the robot path and the position of objects with respect to the robot. Based on this information the software calculates a set of landmark qualitative states and the commands that the robot has to execute to follow the path. The qualitative states are formed using a fuzzy rule base. As a result we have a set of conditions of the form “When an object is in front of the robot and an object is behind right of the robot, turn left”.

To test the accuracy of the set of qualitative conditions with respect to robot navigation along a path, we place the robot in a simulator environment similar to the environment sketched on the PDA. We expect the robot to be able to follow directions given by the set of conditions and navigate to the goal on a similar path to the one sketched on the PDA. Differences between the sketched and the actual (simulated) environment such as scaling, inaccuracies, object shape and even insignificant object existence should not affect the route that the robot executes in the simulator. The idea is: if a human can follow directions on the simulated environment based on the sketch, the robot should be able to do it as well. We have proven by experimentation that indeed this is the case for several simulated maps given a specific sketch. We have yet to test the software on the real robot. To achieve satisfactory results additional features had to be used as part of the set of conditions such as object histogram width, object proximity and temporal data of the spatial relations between the robot and the surroundings. Also, the low level obstacle avoidance module mentioned above has been implemented. This module uses potential fields to account for small deviations in the robot’s heading.

In the future we plan on connecting the spatial behaviors module to the rest of the Guinness modules and to test the whole architecture with a real Nomad200 robot, instead of a simulated environment.

IX. DEMO SCENARIO RESULTS

The following is a log of the input and output of a demonstration using Guinness. Notes in italics explain any actions Guinness takes as a result of the command.

User: Guinness, go home

Guinness: Going home ...

(Guinness moves to a point we have previously defined as home)

User: Guinness, explore here

Guinness: I'm on my way.

(Guinness explores the room by randomly going to a set of 9 point that we have defined within the lab)

User: Guinness, stop

Guinness: stopping

User: Guinness, go home

Guinness: Going home ...

User: Guinness, what do you see

Guinness: There are objects in front of me and on my right, the object number 2 is behind me, and the object number 7 is mostly to the left of me

User: Guinness, where is object 1?

Guinness: right and forward

User: Guinness, how many objects do you see?

Guinness: Looking ...

Guinness: I sense nine objects.

User: Guinness, what objects are on your left?

Guinness: Looking ...

Guinness: The object number 6 is mostly in front of me and the object number 7 is mostly to the left of me

User: Guinness, object 3 is a table

Guinness: Looking ...

Guinness: I now know that object number 3 is a table

User: Guinness, go in front of the table

Guinness: Looking ...

Guinness: Looking ...

(Guinness contacts SRserver to calculate a target point which is in front of the table)

(Guinness moves to the location in front of the table)

User: Guinness, object 6 is Bob

Guinness: I now know that object number 6 is Bob

User: Guinness, where is Bob?
Guinness: right and forward

User: Guinness, go behind Bob
Guinness: Looking ...
(Guinness moves to the location behind Bob)

User: Guinness, face Bob
Guinness: Looking ...
(Guinness turns to face Bob)

User: Guinness, move backwards 1 meter
(Guinness moves back 1 meter)

User: Guinness, do you recognize anything?
Guinness: Looking ...
Guinness: Checking Image Server for ANYTHING.
(ImageServer takes a picture and scans it using the MSNN for Bob and Wiley.)
(ImageServer Log)

```
[2003-05-14 | 14:02:14] <1546> IS_WHAT command received  
[2003-05-14 | 14:02:21] <1546> Send scanned image to screen  
[2003-05-14 | 14:02:29] <1546> Send scanned image to screen  
[2003-05-14 | 14:02:29] <1546> I Found, bob at -3 degrees
```

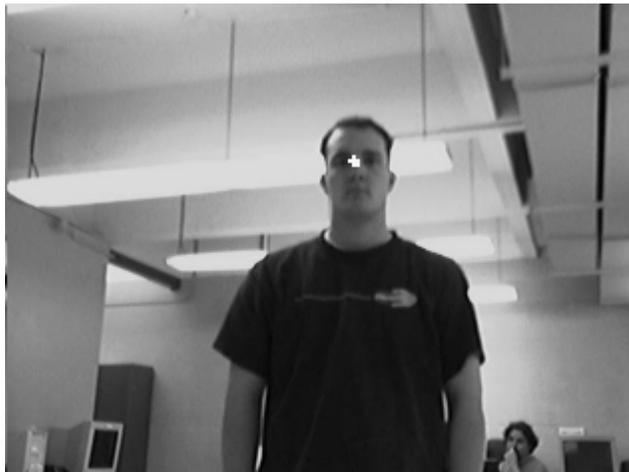


Figure 11
An image taken and processed by Image Server. The white area indicates where Image Server detected Bob.

Example Walkthrough of SRServer output with images.

This version of srserver will try to connect to the robot
if the global variable WITH_WAX is TRUE...

Connecting with Robot...R_Init_Robot failed

NOTICE (Spatial): Setting up as server...

Success.

After setting Signal() trap.

Map loaded from file initial.eg

Init Mygrid...

NOTICE (palmserver): Attempting to connect with Mapserver at
"localhost" port 5012...success!Retrieve map from

mapserver

Map retrieved from mapserver

Last command packet transmitted:

01 02 00 15 18 5c

NOTICE (Spatial): Contacted by a new client!Retrieve map from mapserver and
process

No new map

Last command packet transmitted:

01 02 00 15 18 5c

Object # 1 starts at (79, 18)

Object # 2 starts at (59, 29)

Object # 3 starts at (93, 30)

Object # 4 starts at (96, 58)

Label Objects: 4 objects found

4 Objects labeled

18 points loaded for object# 1

50 points loaded for object# 2

22 points loaded for object# 3

15 points loaded for object# 4

LoadObjectsFromGrid: 4 objects loaded

Objects loaded with 105 scene points

StoreObjdata for 4 objects; Robot at (76,51) Heading = 0.0

Objdata stored for 4 objects

Generate spatial descriptions

OBJECT# 1 starts at 0 with 18 points LABEL: object number 1

Closest point (84,23) at Distance: 29.1 Heading: 285.9

Primary direction: Right Shifted Front Basic8dir: 0xc0 Right Front-right

Assessment: Satisfactory

OBJECT# 2 starts at 18 with 50 points LABEL: object number 2

Closest point (60,50) at Distance: 16.0 Heading: 183.6

Primary direction: Behind Shifted Right Basic8dir: 0x30 Rear Rear-right

Assessment: Satisfactory

OBJECT# 3 starts at 68 with 22 points LABEL: object number 3

Closest point (92,39) at Distance: 20.0 Heading: 323.1

Primary direction: Front Right Basic8dir: 0xc1 Front Right Front-right

Assessment: Satisfactory

OBJECT# 4 starts at 90 with 15 points LABEL: object number 4

Closest point (91,62) at Distance: 18.6 Heading: 36.3

Primary direction: Front Shifted Left Basic8dir: 0x3 Front Front-left

Assessment: Satisfactory

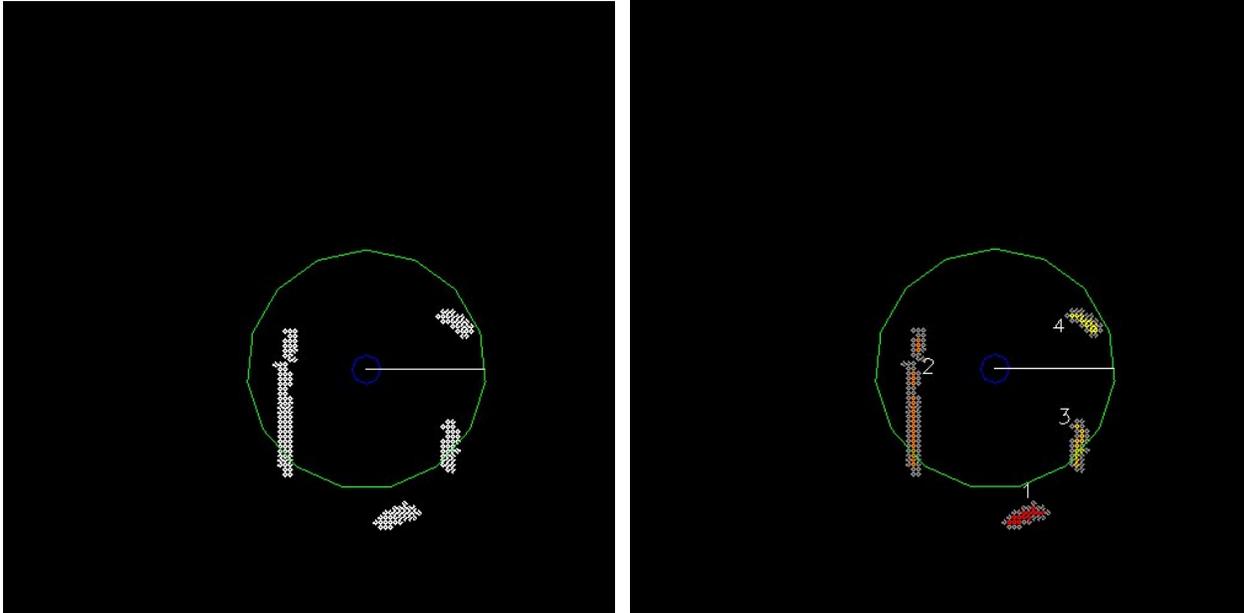


Figure 12 The map loaded by SRserver from MapServer (left) and the segmented objects (right).

DETAILED SPATIAL DESCRIPTIONS for 4 OBJECTS:
 The object number 1 is mostly to the right of me
 but somewhat forward
 (the description is satisfactory)
 The object is close.

The object number 2 is behind me
 but extends to the right relative to me
 (the description is satisfactory)
 The object is very close.

The object number 3 is to the right-front of me
 (the description is satisfactory)
 The object is very close.

The object number 4 is mostly in front of me
 but somewhat to the left
 (the description is satisfactory)
 The object is very close.

Output of SRServer after the “Describe the Scene” command.

HIGH LEVEL DESCRIPTION:

There are objects on my front right.
 The object number 2 is behind me.
 The object number 4 is mostly in front of me.

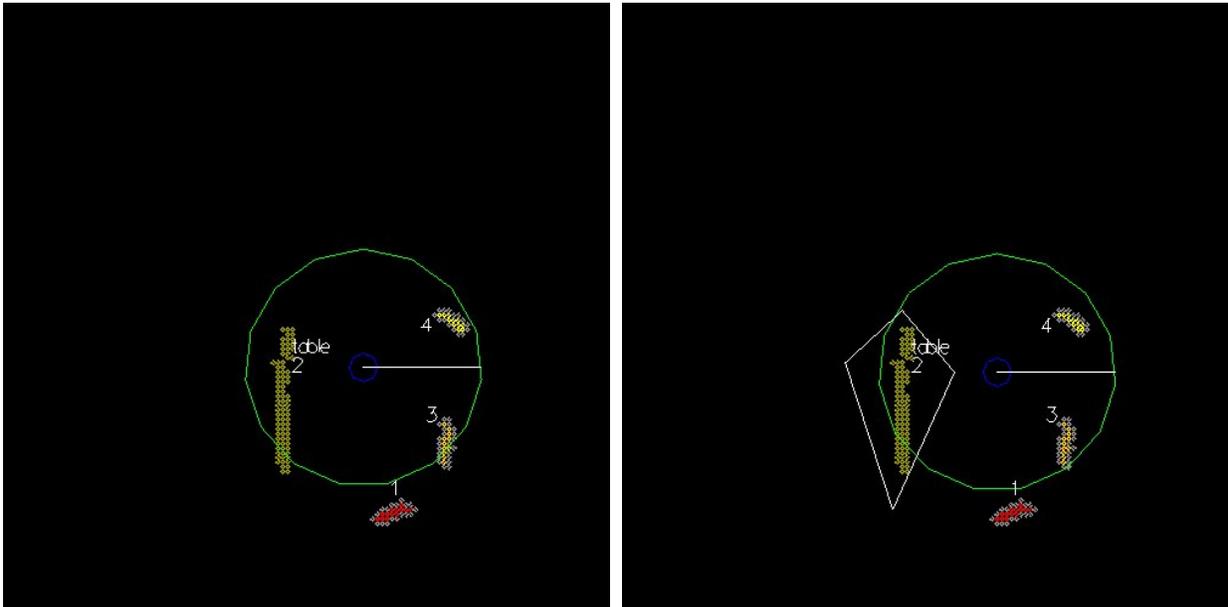


Figure 13 Output of SRServer after labeling object 2 as a "table" (left) and after requesting positions that are left, right, front and behind the table (right) .

```

ASSIGN_LABEL: Objnum = 2
ASSIGN_LABEL: Label = table
ok:0
Number of Persistent Objects is now 1
StorePObjdata for 1 objects; Robot at (76,51) Heading = 0.0
Objdata stored for 1 objects
OBJECT# 1 starts at 0 with 50 points LABEL: table
Closest point (60,50) at Distance: 16.0 Heading: 183.6
Primary direction: Behind Shifted Right Basic8dir: 0x30 Rear Rear-right
Assessment: Satisfactory

```

Persistent Objects:

DETAILED SPATIAL DESCRIPTIONS for 1 OBJECTS:

```

The table is behind me
but extends to the right relative to me
(the description is satisfactory)
The object is very close.

```

WARNING (Spatial): Lost contact with client on read call - freeing slot for new client

NOTICE (Spatial): Trapped ^C, Exiting gracefully.

REFERENCES

- [Chronis 2003] G. Chronis and M. Skubic, "Sketch-Based Navigation for Mobile Robots," in *Proc. of the IEEE 2003 Intl. Conf. on Fuzzy Systems*, May, 2003, St. Louis, MO.
- [Adams 2000] W. Adams, D. Perzanowski, A.C. Schultz, "Learning, Storage and Use of Spatial Information in a Robotics Domain", *Proc. of the ICML 2000 Workshop on Machine Learning of Spatial Language*, Stanford Univ.: AAAI Press, 2000, pp. 23-27.
- [Martin 1996] M.C. Martin, H.P. Moravec, "Robot Evidence Grids", Carnegie Mellon University, Pittsburgh, PA, Technical Report #CMU-RI-TR-96-06, Mar., 1996.
- [Matsakis 1999] P. Matsakis, L. Wendling, "A New Way to Represent the Relative Position of Areal Objects," *IEEE Pattern Analysis and Machine Intelligence*, vol. 21, no. 7, pp. 634-643, 1999.
- [Matsakis 2001] P. Matsakis, J. Keller, L. Wendling, J. Marjamaa, O. Sjahputera, "Linguistic Description of Relative Positions in Images," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 31, no. 4, pp. 573-588, 2001.
- [Perzanowski 1999] D. Perzanowski, A. Schultz, W. Adams, and E. Marsh, "Goal Tracking in a Natural Language Interface: Towards Achieving Adjustable Autonomy," *Proc. of the 1999 IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation*, Monterey, CA, 1999, pp.208-213.
- [Perzanowski 2001] D. Perzanowski, A.C. Schultz, W. Adams, E. Marsh, M. Bugajska, "Building a Multimodal Human-Robot Interface", *IEEE Intelligent Systems*, pp. 16-20, Jan./Feb, 2001.
- [Perzanowski 2002] D. Perzanowski, M. Skubic, A. Schultz, W. Adams, M. Bugajska, K. Wauchope, and E. Marsh, "Multi-modal Navigation of Robots Using Spatial Relations: A Videotaped Demonstration," in *IEEE International Conference on Robotics and Automation 2002 Video Proceedings*, Washington, DC, May, 2002.
- [Schultz 1999] A. Schultz, W. Adams and B. Yamauchi, "Integrating Exploration, Localization, Navigation and Planning with a Common Representation," *Autonomous Robots*, vol.6, no.3, May 1999.
- [Scott 2003] G. Scott, J.M. Keller, M. Skubic and R.H. Luke III, "Face Recognition for Homeland Security: A Computational Intelligence Approach," in *Proc. of the IEEE 2003 Intl. Conf. on Fuzzy Systems*, May, 2003, St. Louis, MO.
- [Skubic 2001a] M. Skubic, P. Matsakis, B. Forrester and G. Chronis, "Extracting Navigation States from a Hand-Drawn Map", in *Proc. of the IEEE 2001 Intl. Conf. on Robotics and Automation*, vol. 1, pp. 259-264, Seoul, Korea, May, 2001.
- [Skubic 2001b] M. Skubic, G. Chronis, P. Matsakis and J. Keller, "Generating Linguistic Spatial Descriptions from Sonar Readings Using the Histogram of Forces", in *Proceedings of the IEEE 2001 International Conference on Robotics and Automation*, vol. 1, pp. 485-490, Seoul, Korea, May, 2001.
- [Skubic 2002a] M. Skubic, S. Blisard, A. Carle, P. Matsakis, "Hand-Drawn Maps for Robot Navigation", AAAI 2002 Spring Symposium, Sketch Understanding, Stanford University, March, 2002 (Technical Report SS-02-08).
- [Skubic 2002b] M. Skubic, D. Perzanowski, A. Schultz, and W. Adams. "Using Spatial Language in a Human-Robot Dialog," in *Proc. of the IEEE 2002 Int. Conf. On Robotics and Automation*, Washington, D.C., May. 2002.
- [Skubic 2002c] M. Skubic and S. Blisard, "Go to the Right of the Pillar: Modeling Unoccupied Regions for Robot Directives", *Technical Report, AAAI Fall Symposium*, Human-Robot Interaction Workshop, Nov., 2002.
- [Skubic 2003a] M. Skubic, P. Matsakis, G. Chronis and J. Keller, "Generating Multi-Level Linguistic Spatial Descriptions from Range Sensor Readings Using the Histogram of Forces", *Autonomous Robots*, Vol. 14, No. 1, Jan., 2003, pp. 51-69.
- [Skubic 2003b] M. Skubic, D. Perzanowski, S. Blisard, A. Schultz, W. Adams, M. Bugajska and D. Brock "Spatial Language for Human-Robot Dialogs," accepted to *IEEE Transactions on SMC, Part C*, special issue on Human-Robot Interaction.

- [Skubic 2003c] M. Skubic, S. Blisard, C. Bailey, J.A. Adams, and P. Matsakis, "Qualitative Analysis of Sketched Route Maps: Translating a Sketch into Linguistic Descriptions," accepted to *IEEE Transactions on SMC, Part B*
- [Wauchope 2000] K. Wauchope, *Eucalyptus: Integrating Natural Language Input with a Graphical User Interface*, Naval Research Laboratory, Washington, DC, Technical Report NRL/FR/5510-94-9711, 2000.
- [Won 1995] Y. Won, *Nonlinear Correlation Filter and Morphology Neural Networks for Image Pattern and Automatic Target Recognition*, Ph. D. Dissertation, University of Missouri – Columbia, 1995.
- [Won 1997] Y. Won, P. Gader, and P. Coffield, "Morphological Shared-Weight Neural Networks with Applications to Automatic Target Recognition," *IEEE Trans. Neural Networks*, Vol. 8, pp. 1195-1203, 1997.

APPENDIX A: HOW TO BRING UP MU-WAX

Phase 1: Powering Up Guinness

- 1) Insert the five batteries that are not on charge (2 18 Amp/hour and 3 12 Amp/hour).
- 2) Power up the robot
- 3) At the lilo-prompt, press enter to load linux-tulipified
- 4) Wait for a few minutes
- 5) Logon as root
- 6) Run "startme.sh"

Phase 2: Powering Up the Laptop

- 1) Connect the webcam to the USB port
- 2) Connect the laptop to the Ethernet switch
- 3) Power up the laptop
 - a. If the "DELL" boot screen doesn't come up, gently thump the laptop directly below the PCMCIA card slots.
- 4) Take "The Red Pill"
- 5) Login as root
- 6) Run "startme.sh"
- 7) Logout as root
- 8) Login as "robot"
- 9) Run "startx"
- 10) Open an xterm
- 11) Enter "konsole &" to spawn a new multi-purpose console
- 12) "su -" in the xterm to go up to root
- 13) in the root-window xterm, enter the following commands
 - a. ./acu
 - b. /etc/init.d/firewall-2.4 start
 - c. ifconfig eth0 192.168.1.1 up

Firing Off MU-WAX

- 1) Place the robot in a known location (i.e. the X by the door)
- 2) telnet 192.168.1.2 65000
 - a. at the robodaemon prompt:
 - b. zr
 - c. turn the robot so the front is pointing into the room with the suns
 - d. dp 1440 0 (places the robot)
 - e. da 0 0 (inits the current heading for the robot to 0,0)
- 3) in the konsole window, enter "./start1"
 - a. when Nserver comes up, select "Real Robot"
- 4) new konsole, run ./start2 (mapserver)
- 5) new konsole, run ./start3 (continuous, wait for connect to the robot)
- 6) new konsole, run ./start4 (vfh)
- 7) new konsole, run ./start5 (trulla)
- 8) new konsole, run ./start6 (palmserver)

- 9) new konsole, enter “export DISPLAY=kronos.cecs.missouri.edu:0.0”
 - a. on kronos, enter into a terminal “xhost +<roboLaptop’s ip>”
 - b. on the laptop, enter “./start7” (imageserver)
- 10) new konsole, enter “./start8” (cortex)
- 11) new konsole, enter “./start9” (gesture)
- 12) on Kronos,
 - a. cd Wax4.5/SR/v1.5.4
 - b. ./srserver
- 13) Fire off nautilus
- 14) Make sure webcam is plugged in if using ImageServer
- 15) Check the status of vfh!! (See #5 below)
- 16) Done!

Things That Can Go Wrong

- 1) Everything: Shoot the robot, shoot the laptop, then yourself, in that order
- 2) Nothing’s talking
 - a. Make sure Ali doesn’t have one of his toys on
 - b. Make sure the Ethernet is plugged in
 - c. Make sure the wireless is up and running (/sbin/ifconfig)
 - d. Make sure that the IP for the laptop is correct
 - e. Make sure the green light on the robot above “Motor” is blinking, if it’s not, then reconnect the keyboard to the robot and enter “./robodaemonrestart.sh” to restart the robot daemon
 - f. Run netstat and check to see which ports are bound up
 - g. Reboot the Laptop. Strange Ethernet behavior has been observed before.
- 3) The laptop freezes
 - a. Press the power button and try to reboot (Hold down for at least 7 seconds)
 - b. If that doesn’t work, hit on the bottom where the PCMCIA cards are inserted (or thereabouts)
 - c. Try to power it back up
 - d. Repeat (b) – (c) until it works
- 4) The laptop died! Like BAD!
 - a. Remove the hard drive
 - b. Insert it into another similar Inspiron 8200
 - c. Keep your fingers crossed
- 5) vfh Keeps Crashing on srserver startup.
 - a. Restart vfh very last to avoid this software’s bugs.