ETL-0455

(2)

IRS: A simulator for autonomous land vehicle navigation

Phillip A. Veatch
Larry S. Davis

Center for Automation Research
University of Maryland
College Park, MD 20742-3411

July 1987

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

Prepared for

U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060-5546

E
T
L

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CAR-TR-310 CS-TR-1889	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION U.S. Army Engineer Topographic Laboratories
6c. ADDRESS (City, State, and ZIP Code) Center for Automation Research College Park, MD 20742-3411	7b. ADDRESS (City, State, and ZIP Code) Fort Belvoir, VA 22060-5546
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL (If applicable) ISTO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DACA76-84-C-0004
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 6230E PROJECT NO.: TASK NO.: WORK UNIT ACCESSION NO.:

11. TITLE (Include Security Classification)

IRS: A Simulator for Autonomous Land Vehicle Navigation

12. PERSONAL AUTHOR(S)

Phillip A. Veatch and Larry S. Davis

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ____ TO N/A	July 1987	61

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

IRS is a computer simulation program that provides a software testbed for autonomous navigation algorithms. The program allows the user to describe a complex world built from spheres, parallelepipeds, planar surfaces, cones, and cylinders. The program simulates the movement of an Autonomous Land Vehicle and constructs video and range images based on the ALV's field of view as the vehicle moves through the world. Ground maps of the world, as perceived by the ALV, are also created.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Rosalene Holecheck	22b. TELEPHONE (Include Area Code) (202) 355-2767 22c. OFFICE SYMBOL ETL-RI-T

Table of Contents

1. Introduction

The Image Range Simulator (IRS) was developed as a tool for the Autonomous Land Vehicle (ALV) project. Ideally, algorithms for processing visual and range images would be developed from real-world data captured by the ALV. However, maintaining an ALV is both expensive and time consuming. Furthermore, changes in weather and the movement of the sun make it very difficult to reproduce conditions exactly for testing purposes. A robot arm carrying a range scanner and video camera that traverses scale model environments has been used to provide an efficient and relatively inexpensive testing ground for navigation programs [Dementhon 1987]. IRS is a computer simulation program that goes beyond mechanical modelling and provides a software testbed for autonomous navigation algorithms by simulating the movement of an ALV and constructing the video and range images that would be in the ALV's field of view as the vehicle moves. The program is based on an image flow simulator described in [Sinha 1984].

An overview of IRS is given in Section 2 along with the results from a typical simulation run. Section 3 is a users' manual for those who wish to use IRS without extensive modifications. Some details of the program's internal code are described in Section 4 as an aid for future hackers.

2. Program Overview

The simulation process in IRS has four major components. First a synthetic world must be specified and a model created. After this initializing step a loop is begun consisting of: 1) creating visual and range images based on the ALV's current location, 2) applying navigation algorithms to determine where the ALV is to move to next, and 3) calculating and then applying a transformation matrix that "moves" the ALV to its next location.

The simulator can model spheres, parallelepipeds, planar surfaces, cones, and cylinders. They can be arbitrarily translated and rotated and may be positioned so that an object is partially or wholly inside of another object (an important property when constructing complex scenes from these basic building blocks). From the user's perspective, the world that the ALV will drive through is specified by a list of objects. Each object consists of a shape (i.e. sphere, cone, etc.) and parameters describing its size, location, and orientation. Inside the simulator, each object consists of an array of surface control points. On a cone, for example, the control points are the tip of the cone and several equally spaced points on the rim of the cone's base. The centroid of an object is initially placed at the origin of the coordinate system and the locations of its surface points are set according to its shape and size. A transformation matrix is calculated that "moves" the object from the origin to its location and orientation in the world. The object is then positioned by multiplying its control points by this matrix.

After each object has been positioned a visual image is calculated based on a perspective projection in which the focal point is at the origin of the coordinate system and the image plane is placed in front of it at $z =$ focal length. The focal length and the field of view are parameters that the user provides at the start of the program. These parameters, and all other input to the program, can either be read from a file or entered interactively in response to prompts.

The visual image is created by breaking an object's surface into triangles. This triangulation obviously decreases the accuracy of the range image for curved surfaces but any desired level of accuracy can be achieved by increasing the number of control points.

An intensity value is calculated for the center of each triangle and all points within the triangle are assumed to have the same intensity. This assumption leads to artifacts in the visual image. Section 4.5 discusses how to remedy this.

The gray levels can be created with the light source at any position. Surface reflectance is assumed to be Lambertian and all objects have the same albedo (it would be a simple extension to add varying albedos). No compensation is made for reduction in intensity due to increased distance from the light source.

The vertices of each triangle are projected into the image plane and pixels within the projected triangle are all given the same gray level. At first, pixels were assigned z values based on simple interpolation of the z values of the three projected vertices. However, linear interpolation between rows of an image was found to be too inaccurate. Instead, pixels on the edge of the triangle in each row of the image are projected back out to the object and their actual z values are calculated. Within a row, linear z interpolation between the two edge pixels is usually sufficient. Hidden surfaces are removed by comparing z values at each pixel and choosing the surface that has the minimum value.

From the visual image and the corresponding z values we can create a "equirectangular" range image whose pixels are spaced at equal linear intervals on the image plane. However, the ERIM range scanner produces images that are at equal angular intervals on the image plane, so the range image is resampled to accurately simulate the ALV's range scanning process. Interpolation of the range image is done using an intentionally crude algorithm to introduce noise into the system (triangulating and digitizing the image of the objects has already introduced some noise). The final "equiangular" range image has all of the properties of an image produced by an ERIM scanner mounted on an ALV including the same field of view, eight bit range values, and 64 foot ambiguity intervals.

Once the range image is created, the program's modularity allows the use of any navigation algorithms to determine to where the ALV should move. In the program's current configuration the range derivative algorithm, described in [Veatch 1987], is applied to the equiangular range image and the resultant binary obstacle image is mapped from spherical coordinates into the Cartesian xz ground plane. The ground plane map initially has four types of pixels: 1) traversable terrain, 2) obstacles or unnavigable terrain, 3) areas whose traversability is unknown because they are hidden by an obstacle (i.e. shadow regions), and 4) areas whose traversability is unknown because they are outside of the field of view of the simulated range sensor. The path planner will treat the ALV as if it were the size of a single pixel so a boundary the width of the ALV's radius is grown around all obstacle and shadow pixels.

Each pixel in a ground plane map corresponds to one square foot and the entire map covers approximately 65,000 square feet. The vehicle is always at the center of the current map. In addition to the regions seen in the most recent range image, the current map also contains information gathered from previous images and projected into the current coordinate system's ground plane.

At the start of the simulation the program requests the coordinates of the ultimate goal for the ALV. A straight line from the current location to this goal is plotted and a move along it is calculated. The endpoint of the move is passed to the path planner which tries to find a path through the ground map from the current location to the endpoint. The path planner was developed by Kambhampati and Davis and is described in [Kambhampati 1986]. It uses a hierarchical algorithm based on a quadtree division of the ground map. The planner assumes that the vehicle can only travel through pixels that are marked as traversable. [Puri 1987] describes an advanced version of this planner that determines when the vehicle should try to move to a different vantage point so as to see if shadow regions are actually traversable. This can significantly improve the vehicle's path when tall obstacles obscure large regions.

If the planner fails to find a path to the first endpoint a set of heuristics are used in sequence to select alternate subgoal locations. Subgoals are passed, one at a time, to the path planner until one is found that can be reached. If all of the heuristics are exhausted without a reachable subgoal being found, the program notifies the user and gracefully terminates.

Once the endpoint of the next move is found a transformation matrix is calculated that will place the origin of the coordinate system at this new location. This matrix, when applied to each object's control points, will result in the next visual and range images being what the ALV would see if it were driven to the endpoint. The matrix is constructed so that the vehicle will be facing the ultimate goal location (other constraints on what direction the vehicle should be facing or how long each move should be are adjustable parameters in the program). If the move's endpoint is the same as the goal location the program terminates. Otherwise the transformation matrix is applied, the new visual image is found and the program begins another pass at moving the simulated vehicle toward its goal.

A typical trip by the ALV through synthesized terrain is illustrated in Figures 1–9. The visual images at the start of each move are shown in Figure 1. The equirectangular range image at the start of the trip is given in Figure 2. It corresponds to the visual image labelled Time 0 in Figure 1. A montage of the equiangular range images is presented in Figure 3. The four scenes in the montage, in order from top to bottom, are from Times 0, 1, 2, and 3. Figure 4 shows the obstacle pixels found in each equiangular range image. These pixels are mapped into the ground plane in Figures 6–9. The solid black regions in the ground maps are obstacles. White areas are navigable terrain. Horizontal stripes are shadow regions while vertical stripes delimit the grown boundaries surrounding obstacles and shadows. Regions outside of the range scanner's field of view are gray. A key to these markings is provided in Figure 5.

3. User's Manual

This section describes the details necessary to use IRS in its current format. IRS has its origin in three programs written for unrelated projects: 1) an image flow simulator called IFS [Sinha 1984]; 2) a collection of algorithms for detecting obstacles in range image [Veatch 1987]; and 3) a quadtree path planning program [Kambhampati 1986]. These programs were linked with the minimum number of alterations. As a result, the user has to contend with several parameters that must be properly set but which have no obvious meaning in the current version of IRS.

The world coordinate system used in IRS is shown in Figure 10. The user is standing at the origin of the system and looking at the image plane on the Grinnell display. The positive x axis is therefore on the left side of the screen from the user's viewpoint. The image plane is centered on the z axis at z = focal length. If the size of the image plane is given as L then the upper left corner of the image plane will be at ($L/2$, $L/2$, focal_length) and the lower right corner of the image plane will be at ($-L/2$, $-L/2$, focal_length). The term "image plane" is used loosely here to mean the square portion of the infinite image plane that is within the user's field of view. Figure 10 also shows how a world point P is projected onto the image plane at point p. The focal length and the size of the image plane are parameters that the user is prompted for by the program.

IRS has four basic object shapes: cone, cylinder, parallelepiped, sphere. In the following section, an annotated transcript of a typical IRS run is given which includes examples of each object type. Before reading the transcript, several conventions should be understood:

1) For each object, a location is given by the user. This is the location of the object's centroid for cylinders, parallelepipeds, and spheres. The location of a cone is specified by giving the location of the center of the cone's base.

2) IRS is also capable of drawing rectangular planar patches. Whenever a user requests that IRS create a parallelepiped, the program asks, "Do you want to treat the cube as a planar surface?". If the user replies yes, the program creates a parallelepiped in which only the bottom face of the parallelepiped is actually used in the scene. Note that the location the user gives is still the centroid of the full parallelepiped. See line 104 in the transcript for an example of a rectangular planar patch.

3) IRS has a menu which includes an object shape called "surface". This shape is allegedly an arbitrary second order polynomial restricted to an area near the center of the field of view. This is a questionable feature. The original IFS authors gave an example of a "surface" that was simply a planar patch. Recent experiments with full second order surfaces have resulted in objects that appear to be incorrectly drawn. The user is advised to use the planar surface option of a "cube" shape and avoid "surface".

4) At any prompt that requires a "yes" or "no" answer, the user can use "y" or "n". In fact, any string that begins with "y" or "n" will be accepted.

5) IRS expects all size and location values to be in floating point format. However, as the transcript shows, small integer values are read correctly. Rota-

tion values, on the other hand, must be in integer format.

6) The command line for IRS contains either three or four arguments, i.e. "IRS camera_parameters scene_parameters curve_file debug_flag". The first two arguments are filenames that contain information used by the program when it is not in interactive mode. IRS can be run with varying degrees of interaction. The transcript shows the full interactive mode. However, if at line 35, the user answered "no" to the question, "Do you wish to set program parameters interactively?", then camera_parameters must be a file that contains the replies that are given in lines 6–48 (each reply must be on a separate line in the file). Similarly, if the question on line 50, "Do you want to create the scene's objects interactively?", is answered affirmatively then all of the replies from lines 54–525 must be on separate lines in scene_parameters.

If the program does not use these files, there must still be a string in their place on the command line. However, unused files are not opened so any string can be written on the command line.

Curve_file is a leftover from IFS and is never used by IRS so any string can be used for it. If a fourth argument is present and its first character is "d" then certain debugging information is printed at run-time. This is another leftover that is of little use to IRS users.

The following is the transcript of an IRS run. This run produced the images shown in Figures 1–9 of Section 2. The numbered lines are from the actual transcript. Comments about the transcript begin with "==>". The user responses are underlined.

1 1 C: IRS viewing.parameters object.parameters dummy.file

==> In the following demonstration, the three parameters files will not be used but, as explained earlier, they must still be given on the command line.

2 Image Range Simulator [version 1.0]

3 Do you need help? no

==> If you answer yes, a short description of the coordinate system and the image plane is printed. The last sentence in the description refers to "velocities". This is leftover from the image flow program and is no longer pertinent.

4 Do you want to set program parameters interactively?

5 yes

==> If you answer no, then the viewing parameters file is read for all of the answers from lines 6–48. All of the questions are still printed on the screen but the answers are not echoed.

6 Do you want debugging statements executed?

7 no

==> Another leftover from IFS. Just say no.

8 Do you want objects to have independent motion?

9 no

==> IFS allegedly had the capacity to give each object independent motion. This code is still in the program but it isn't well-tested. It probably does not work.

10

11 Setting up the Grinnell display window parameters -

12 Enter Grinnell window size(integer): 255

==> The square field of view on the image plane is projected into a square area on the Grinnell. The value entered here determines the size of the Grinnell picture (in this case, 255 pixels × 255 pixels).

13 Enter the coordinates of the lower left hand corner

14 of the Grinnell window to be used.

15 Column value = 256

16 Row value = 0

==> These coordinates dictate where on the Grinnell screen the image will appear. Note that (column=0, row=0) is the lower left corner of the Grinnell screen (IRS and IFS were written before the DAP package standardized the coordinate system to have its origin in the upper left corner of the screen).

Grinnell opened.
Grinnell cleared.

Setting up the observer's camera parameters -

Enter focal length of camera unit(typically 1): .479

==> The focal length is the z coordinate of the focal plane. See Figure 10.

Enter image plane size(typically 1): 1.0

Are you are producing a stereo image?
Your reply must be either yes or no
no

==> A leftover from IFS. You must answer no.

Set algorithm type -
Options:
0: Fast algorithm, light source fixed at origin.
1: Light source position variable
Choice: 0

==> If the user replies "1", IRS will ask the position of the light source. In addition, whenever the current visual image is displayed on the Grinnell, a prompt will ask if the light position should be changed.

Set the viewer motion parameters -
Translational velocities in units/time step
Vx = 0
Vy = 0
Vz = 0

Rotational velocities in radians/time step
Ox = 0
Oy = 0
Oz = 0
Specify maximum simulation time steps: 0

==> The translational velocities, rotational velocities, and time steps are all leftovers from IFS. Just answer 0.

50 Do you want to create the scene's objects interactively?
51 your reply must be either yes or no
52 yes
53
54 Set up the scene -
55 Menu:
56 Choose objects in scene(up to 5 of any one type)
57 0 ---> to terminate object creation loop
58 1 ---> Cone
59 2 ---> Cylinder
60 3 ---> Parallelepiped
61 4 ---> Sphere
62 5 ---> Surface
63 6 ---> Help function
64 Choice of object number: 3
65
66 Parallelepiped is located (0,0,0),(length,0,0)
67 (0,0,breadth),(0,height,0).......
==> Lines 66 and 67 make no sense at all. They are printed whenever the user chooses shape 3. Ignore them.
68 Length of parallelepiped(in x) = 224
69
70 Breadth of parallelepiped(in z) = 4
71
72 Height of parallelepiped(in y) = 8
73
74 Do you wish to treat cube as a planar surface? Your reply must be either yes or no
75 no
76 Euler angle of parallelepiped (in integer degrees).
77 Rotation about x(horizontal) axis = 0
78 y(vertical) axis = 0
79 z(horizontal) axis = 0
==> Rotations are done first about the x axis, then the y axis, and finally about the z axis.
80
81 Where would you like to place the parallelepiped?
82 Enter the x-coordinate of the parallelepiped origin 230
83 Enter the y-coordinate of the parallelepiped origin -31
84 Enter the z-coordinate of the parallelepiped origin 420
85
86
87 You have the option of seeing the scene from the
88 observer's point of view or from another point in space
89

90 Will you see observer's view? Your reply must be either yes or no
91 yes
==> IRS was written to model the movement of an ALV that is always at the origin of the current coordinate system so this question is always answered affirmatively. If the user replies "no", the program will prompt for the new point of view. Keep in mind that changing the visual image's viewpoint will have the same effect on the range image.
92 Rectangular parallelepiped drawn
93
94 Delete object from scene?
95 no
==> If you do not like the image on the Grinnell, you can delete the object you just created.
96 Choose objects in scene(up to 5 of any one type)
97 0 ---> to terminate object creation loop
98 1 ---> Cone
99 2 ---> Cylinder
100 3 ---> Parallelepiped
101 4 ---> Sphere
102 5 ---> Surface
103 6 ---> Help function
104 Choice of object number: 3
105
106 Parallelepiped is located (0,0,0),(length,0,0)
107 (0,0,breadth),(0,height,0).......
108 Length of parallelepiped(in x) = 1600
109
110 Breadth of parallelepiped(in z) = 1300
111
112 Height of parallelepiped(in y) = 2
113
114 Do you wish to treat cube as a planar surface? Your reply must be either yes or no
115 yes
==> This is an example of using a cube to create a planar patch.
116 Euler angle of parallelepiped (in integer degrees).
117 Rotation about x(horizontal) axis = 0
118 y(vertical) axis = 0
119 z(horizontal) axis = 0
120
121 Where would you like to place the parallelepiped?
122 Enter the x-coordinate of the parallelepiped origin 300
123 Enter the y-coordinate of the parallelepiped origin -37.0
124 Enter the z-coordinate of the parallelepiped origin 600

You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
yes
Planar surface drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 --> to terminate object creation loop
1 --> Cone
2 --> Cylinder
3 --> Parallelepiped
4 --> Sphere
5 --> Surface
6 --> Help function
Choice of object number: 3

Parallelepiped is located (0,0,0),(length,0,0)
(0,0,breadth),(0,height,0).......
Length of parallelepiped(in x) = 26

Breadth of parallelepiped(in z) = 10

Height of parallelepiped(in y) = 14

Do you wish to treat cube as a planar surface? Your reply must be either yes or no
no
Euler angle of parallelepiped (in integer degrees).
Rotation about x(horizontal) axis = 0
y(vertical) axis = 0
z(horizontal) axis = 0

Where would you like to place the parallelepiped?
Enter the x-coordinate of the parallelepiped origin -90.0
Enter the y-coordinate of the parallelepiped origin -25.5
Enter the z-coordinate of the parallelepiped origin 120

You have the option of seeing the scene from the
observer's point of view or from another point in space

```
Will you see observer's view?  Your reply must be either yes or no
yes
Rectangular parallelepiped drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 3

Parallelepiped is located (0,0,0),(length,0,0)
(0,0,breadth),(0,height,0).......
Length of parallelepiped(in x)  = 9

Breadth of parallelepiped(in z) = 6

Height of parallelepiped(in y)  = 9

Do you wish to treat cube as a planar surface?  Your reply must be either
yes or no
no
Euler angle of parallelepiped (in integer degrees).
  Rotation about   x(horizontal) axis = 0
                   y(vertical) axis = 0
                   z(horizontal) axis = 0

Where would you like to place the parallelepiped?
Enter the x-coordinate of the parallelepiped origin -72.5
Enter the y-coordinate of the parallelepiped origin -29.0
Enter the z-coordinate of the parallelepiped origin 120


You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view?  Your reply must be either yes or no
yes
Rectangular parallelepiped drawn
```

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 2

Cylinder is drawn from +length/2 to -length/2
Length of cylinder = 10

Radius of cylinder = 1.5
Euler angle of cylinder (in integer degrees).
Rotation about x(horizontal) axis = 90
y(vertical) axis = 0
z(horizontal) axis = 0

Where would you like to place the cylinder?
Enter the x-coordinate of the cylinder origin -98.0
Enter the y-coordinate of the cylinder origin -33.5
Enter the z-coordinate of the cylinder origin 120

You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
yes
Cylinder drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function

```
Choice of object number: 2

Cylinder is drawn from +length/2 to -length/2
Length of cylinder = 7

Radius of cylinder = 1.5
Euler angle of cylinder (in integer degrees).
  Rotation about  x(horizontal) axis = 90
                  y(vertical) axis = 0
                  z(horizontal) axis = 0

Where would you like to place the cylinder?
Enter the x-coordinate of the cylinder origin -71.0
Enter the y-coordinate of the cylinder origin -33.5
Enter the z-coordinate of the cylinder origin 120


You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view?  Your reply must be either yes or no
yes
Cylinder drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 1

Cone origin is at center of base
Height of cone = 9

Radius of cone = 7
Euler angle of cone (in integer degrees).
  Rotation about  x(horizontal) axis = 0
                  y(vertical) axis = 0
                  z(horizontal) axis = 0
==>       The rotation of a cone is slightly different than that of other
```

objects. All other objects are rotated about their centroids, which is fairly intuitive. A cone, however, has its origin in the center of its base. Rotation is done about axes whose origin is at the center of the base of the cone.

Where would you like to place the cone?
Enter the x-coordinate of the cone origin 0
Enter the y-coordinate of the cone origin -36
Enter the z-coordinate of the cone origin 120

You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
yes

*****Warning: Unstable solution in find_z at row,col = (154, 127)

==> An unstable numerical solution has been detected while projecting a control point onto the image plane. The solution is still usually adequate. The location given is the row and column in the original range image where the problem occurred (note that the row and column in this error message is based on (0,0) being in in the upper left corner. For more details see Section 4.6.

*****Warning: Unstable solution in find_z at row,col = (161, 127)

*****Warning: Unstable solution in find_z at row,col = (161, 127)
Cone drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 1

Cone origin is at center of base
Height of cone = 8

Radius of cone = 7
Euler angle of cone (in integer degrees).
  Rotation about x(horizontal) axis = 180
                  y(vertical) axis = 0
                  z(horizontal) axis = 0

Where would you like to place the cone?
Enter the x-coordinate of the cone origin 50
Enter the y-coordinate of the cone origin -28
Enter the z-coordinate of the cone origin 70

You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
yes

*****Warning: Unstable solution in find_z at row,col = (190, 40)

*****Warning: Unstable solution in find_z at row,col = (181, 30)

*****Warning: Unstable solution in find_z at row,col = (181, 30)
Cone drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 4

Center of sphere is at origin
Radius of sphere = 25

Euler angle of sphere (in integer degrees).
Rotation about x(horizontal) axis = 0
y(vertical) axis = 0
z(horizontal) axis = 0

Where would you like to place the sphere?
Enter the x-coordinate of the sphere origin 115
Enter the y-coordinate of the sphere origin -366
Enter the z-coordinate of the sphere origin 590

You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
yes
Sphere drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 3

Parallelepiped is located (0,0,0),(length,0,0)
(0,0,breadth),(0,height,0).......
Length of parallelepiped(in x) = 20

Breadth of parallelepiped(in z) = 20

Height of parallelepiped(in y) = 20

Do you wish to treat cube as a planar surface? Your reply must be either yes or no
no
Euler angle of parallelepiped (in integer degrees).
Rotation about x(horizontal) axis = 45
y(vertical) axis = 0
z(horizontal) axis = 45

```
Where would you like to place the parallelepiped?
Enter the x-coordinate of the parallelepiped origin 270
Enter the y-coordinate of the parallelepiped origin -36
Enter the z-coordinate of the parallelepiped origin 520



You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
your reply must be either yes or no
yes
Inaccurate estimate for z along edge at row= 120, col= 191
Rectangular parallelepiped drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 2

Cylinder is drawn from +length/2 to -length/2
Length of cylinder = 15

Radius of cylinder = 5
Euler angle of cylinder (in integer degrees).
  Rotation about    x(horizontal) axis = 0
                    y(vertical) axis = 0
                    z(horizontal) axis = 0

Where would you like to place the cylinder?
Enter the x-coordinate of the cylinder origin 250
Enter the y-coordinate of the cylinder origin -29
Enter the z-coordinate of the cylinder origin 700


You have the option of seeing the scene from the
```

observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
yes
Cylinder drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number: 2

Cylinder is drawn from +length/2 to -length/2
Length of cylinder = 15

Radius of cylinder = 5
Euler angle of cylinder (in integer degrees).
Rotation about x(horizontal) axis = 0
y(vertical) axis = 0
z(horizontal) axis = 0

Where would you like to place the cylinder?
Enter the x-coordinate of the cylinder origin 350
Enter the y-coordinate of the cylinder origin -29
Enter the z-coordinate of the cylinder origin 700

You have the option of seeing the scene from the
observer's point of view or from another point in space

Will you see observer's view? Your reply must be either yes or no
yes
Cylinder drawn

Delete object from scene?
no
Choose objects in scene(up to 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone

516 2 ---> Cylinder
517 3 ---> Parallelepiped
518 4 ---> Sphere
519 5 ---> Surface
520 6 ---> Help function
521 Choice of object number: 0
522
523 Would you like to define feature points on the image?
524 Your reply must be either yes or no
525 no

==> "Feature points" were used in IFS. They basically allow the user to draw features on an object. They have not been tested with IRS but are available for the adventurous. See [Sinha 1984] for a full description of feature points.

526
527 Equiangular and flatworld frame values are being initialized with standard values

==> These values were set to model an ERIM range scanner and to meet the requirements of the path planner. Section 4.2 tells how to alter them.

528 Vel: $x=$ 0.000000 $y=$ 0.000000 $z=$ 0.000000
529 Vel: $x=$ 0.000000 $y=$ 0.000000 $z=$ 0.000000
530 Vel: $x=$ 0.000000 $y=$ 0.000000 $z=$ 0.000000
531 Cumulative Transformation Matrix
532 1.000000 0.000000 0.000000 0.000000
533 0.000000 1.000000 0.000000 0.000000
534 0.000000 0.000000 1.000000 0.000000
535 0.000000 0.000000 0.000000 1.000000
536 Instantaneous OMTM Transform
537 1.000000 0.000000 0.000000 0.000000
538 0.000000 1.000000 0.000000 0.000000
539 0.000000 0.000000 1.000000 0.000000
540 0.000000 0.000000 0.000000 1.000000
541

==> The program has completed the object formation stage and is about to draw the world as the ALV will see it before the ALV moves anywhere. Every time the world is drawn, the transformation matrices shown in lines 531–535 and 536–540 are printed. The Cumulative matrix is a leftover from IFS. In IRS, the Cumulative matrix and the Instantaneous matrix have the same value. Section 4.3 discusses these matrices in more detail. They currently are equal to the identity matrix because the ALV has not yet moved.

The velocities on lines 528–530 are also IFS leftovers.
They are always zero.

542
543 *****Warning: Unstable solution in find_z at row,col = (154, 127)
544
545
546 *****Warning: Unstable solution in find_z at row,col = (161, 127)
547
548
549 *****Warning: Unstable solution in find_z at row,col = (161, 127)
550 Cone drawn
551
552
553 *****Warning: Unstable solution in find_z at row,col = (190, 40)
554
555
556 *****Warning: Unstable solution in find_z at row,col = (181, 30)
557
558
559 *****Warning: Unstable solution in find_z at row,col = (181, 30)
560 Cone drawn
561 Cylinder drawn
562 Cylinder drawn
563 Cylinder drawn
564 Cylinder drawn
565 Rectangular parallelepiped drawn
566 Planar surface drawn
567 Rectangular parallelepiped drawn
568 Rectangular parallelepiped drawn
569 Inaccurate estimate for z along edge at row= 120, col= 191
570 Rectangular parallelepiped drawn
571 Sphere drawn
572
573 Save final visual scene from this pass?
574 yes

==> IRS allows the user to save a variety of images during the simulation. Whenever the user answers affirmatively, a filename is requested. The image is saved in cvl picture file format in the directory that the user is currently in.

575
576 Enter the filename in which to save: visual.time0
577 ***Warning: negative range(=–44) at r=135, c=66 in saverange()
578 ***Warning: negative range(=–44) at r=135, c=67 in saverange()

==> IRS has a bug in it. When triangles are projected onto the image plane, round-off will sometimes leave a pixel without any value.

When modelling an ERIM scanner, however, this bug is actually a feature since it emulates a problem that the scanner has with producing actual range images. This is why it was left in IRS. If your obstacle algorithms cannot handle a few gross position errors, the algorithms will not work on real range data.

579
580 Save the range image?
581 Your reply must be either yes or no
582 yes
583
584 Enter filename in which to save range image: range.equiangular.time0
585
586 What are the x ($+x$ to the left) and z coordinates of the goal (floating point)?
587 300.0 700.0

==> This is the ultimate location that the ALV is trying to reach.

588
589 Do you wish to save equiangular range? Your reply must be either yes or no
590 yes
591 then enter filename: range.time0
592
593 Shall all thresholding be done using automatic cutoffs?
594 Your reply must be either yes or no
595 yes

==> These are the thresholds used by the obstacle detection algorithms. Appendix C explains how to change the automatic cutoff values. If you don't want to use automatic levels, the program allows you to pause and threshold the images manually before continuing. Appendix C contains an example of this.

596
597 Do you wish to save obstacle array? no

==> The obstacle array contains a binary image in which non-zero pixels are obstacles. The array was produced by running the obstacle detection algorithms on the equiangular range image. Figure 4 shows a montage of four obstacle images (these images, of course, were made in an earlier run in which the user answered "yes" to the prompt on line 597).

598
599 Do you wish to save flatw after integrate? yes
600 then enter filename: flatworld.time0

==> "flatw" is short for "flat world". This is another name for the ground plane map that is described in Section 2. It is the projection of the equiangular range image onto the Cartesian xz plane. As described in Section 2, the projection has four pixel types: traversable, obstacles, hidden, and out-of-view.

601

602 Do you wish to save depth after integrate? Your reply must be either yes or no

603 no

==> "depth" is an image that contains the z value for each pixel in flatw.

604

605 Do you wish to save flatw after grow? yes

606 then enter filename: grown.flatworld.time0

==> "flatw after grow" is the flatw image with the addition of a boundary grown around each obstacle and hidden pixel as described in Section 2.

607 Shall the binary map for the path planning routine be placed in

608 the file <binary_map>? (y/n) Your reply must be either yes or no

609 yes

==> A negative response would cause IRS to prompt the user to name the file that the map should be placed in. Appendix A contains a complete explanation of what this map looks like and how to use it for path planning.

610 The binary map for the path planner is in the file <binary_map>

611 The start_node is (128, 127) and the goal_node is (104, 182)

612 Type <control-z> to put this process to sleep and to allow

613 you to run the Puri Path Planning routine.

614 After the path planner is done and you have restarted this

615 program, type <yes> to continue program

616 ^z

==> If the operating system does not allow you to suspend a program by typing <control-z> or some other signal, then IRS will have to be modified to permit this interruption.

617 Stopped

618 2 C: run.path.planner

==> "run.path.planner" is a shell file that runs the path planning routine described in Appendix A. The transcript of the routine is not included here because it is quite lengthy and uninformative. It is anticipated that future users will probably use some other method for path planning.

619

620 3 C: fg

621 IRS viewing.parameters object.parameters dummy.file

622 yes

623

624 Did path planner find a path? yes

```
Next Transformation Matrix:  0.9203 0.0000    0.3911 0.0000
                             0.0000 1.0000    0.0000 0.0000
                            -0.3911 0.0000    0.9203 0.0000
                            -2.3008 0.0000 -240.0223 1.0000

Goal Coordinate After Transform= (0.00, 0.00, 521.55)
```

==> This is the location of the ALV's ultimate goal in the new world coordinate system. See Section 4.3 for details.

```
Vel: x= 0.000000 y= 0.000000 z= 0.000000
Vel: x= 0.000000 y= 0.000000 z= 0.000000
Vel: x= 0.000000 y= 0.000000 z= 0.000000
Cumulative Transformation Matrix
 0.920331  0.000000    0.391141  0.000000
 0.000000  1.000000    0.000000  0.000000
-0.391141  0.000000    0.920331  0.000000
-2.300827  0.000000 -240.022304  1.000000
Instantaneous OMTM Transform
 0.920331  0.000000    0.391141  0.000000
 0.000000  1.000000    0.000000  0.000000
-0.391141  0.000000    0.920331  0.000000
-2.300827  0.000000 -240.022304  1.000000
Cone drawn
Cone drawn
Cylinder drawn
Cylinder drawn
Cylinder drawn
Cylinder drawn
Rectangular parallelepiped drawn
Planar surface drawn
Rectangular parallelepiped drawn
Rectangular parallelepiped drawn
Rectangular parallelepiped drawn
Inaccurate estimate for z along edge at row= 120, col= 89
Inaccurate estimate for z along edge at row= 115, col= 88
Inaccurate estimate for z along edge at row= 107, col= 85
Inaccurate estimate for z along edge at row= 110, col= 83
Sphere drawn

Save final visual scene from this pass?
```

==> This is the start of the second cycle of the run (i.e. Time 1). It proceeds exactly the same as the Time 0 cycle except that the final goal is not requested again. If the next move would place the ALV at the ultimate goal location, the program terminates. This is described in more detail in Section 4.3.

4. A Hacker's Guide to IRS

IFS has about 10,000 lines of C source code. It is expected that all but the most casual users will need to modify the program in some way to meet their particular needs. This section provides the user with some insight into the program's structure as well as directions for modifying certain functions of the program. Earlier sections have described what IRS does; this section describes what functions and files in the source code actually perform specific tasks.

4.1. Program Structure

IRS has been previously described as being the result of combining two parts: an image flow simulator called IFS and a collection of range image navigation programs. The functions navigate() and frame_initialize() contain the range navigation routines while all of the other code called by main() comes from IFS. A rough outline of IRS's structure is:

```
main()
 {
  [initialize visual camera parameters];
  c_scene(); /* create objects */
  frame_initialize(); /* initialize the "frame" variable (more on this later) */
  while (not_done)
   {
    c_process(); /* Apply transformation matrix to control points and
                    form visual and equirectangular range images */
     navigate()
      {
       make_equiangular(); /* form ERIM range image */
       detect_obstacles(); /* find obstacles in range image */
       make_flat(); /* form ground map */
       find_path(); /* find a path through ground map */
       make_new_transform(); /* calculate matrix for moving ALV */
      }
   } /* end while-loop */
```

```
}     /* end main() */
```

The two parts of IRS have very different flavors to their programming style. In particular, IFS extensively uses global variables while the range navigation functions do not. The global variables, compile-time constants, and common data structure declarations for IFS routines are kept in the file prog.h. Constants and common data structure declarations for the range navigation routines are kept in irs.h. Constants, data structures, and basic system #include files that are needed by all IRS functions are in prog.irs.h.

A few files require prog.irs.h, irs.h, *and* prog.h. Whenever this is necessary, prog.h must come before irs.h, and irs.prog.h is not explicitly included because it will be added recursively by prog.h (comments in the irs.h file explain this in more detail).

Experienced C programmers will notice that global variables in IRS are created in a way that violates how C is suppose to work. In theory, only one file should contain the global variables' definitions and all other files that use the global variables should have only external declarations. In practice, prog.h (which contains only definitions) is #include'd in each of the files that need to access its variables. This should result in each file having variables that are global to the individual files but *not* global to the other files. IRS, as it is currently written, runs correctly when compiled with the standard cc compiler supplied with BSD 4.2 and 4.3. To make IRS conform to standard C, one should simply copy the global variable definitions into a file and replace the definitions in prog.h with "extern" declarations.

The IFS code and the range navigation code also differ in their internal world coordinate systems. The user has to know the IFS coordinate system (shown in Figure 10) because it is the system used to specify where objects are located and the location of the ALV's ultimate goal. However, if one wishes to understand the range navigation code it is necessary to realize that much of it is based on the range scanner coordinate system shown in Figure 11. In this system, the positive y axis is pointing down from the camera/range scanner toward the ground and the positive x axis is in the opposite direction of IFS's x axis. Section 3 in [Veatch 1987] gives a complete description of the range scanner coordinate system and how it is related to the equiangular range image array.

4.2. Changing Image Parameters

All of the parameters for the visual scene in IRS are initialized at the start of each run by the user. The annotated run shown in Section 3 shows how this is done and describes how the parameters can be placed in a file for reuse. The prompts for these visual parameters are given in main(), c_algorithm(), and c_scene(). The parameters' values are stored in global variables defined in prog.h.

The range navigation portion of IRS avoids storing parameters in global variables by keeping many of them in a variable called "frame". Frame contains the parameters for three images: the equirectangular range image, the equiangular range image, and the flat-world image (or ground map). Frame is initialized by the function frame_initialize(). These values were not expected to change frequently so they are kept as compile-time constants in the init_frame.c file (frame_initialize() is also in this file). If an application requires that they be changed frequently, it would be a trivial matter to have frame_initialize() prompt the user instead of using constants. The data structure (called frame_data) of the frame variable is declared in prog.irs.h.

Although the fields of frame_data are explained in prog.irs.h, the flat-world parameters need further explanation. The variable "flatw" is conceptually a map that the simulated ALV uses to drive through its world. To simplify navigation, the map is two-dimensional (i.e. if the range image contains a pixel corresponding to some (x,y,z) that is determined to be an obstacle then the pixel in flatw corresponding to (x,z) is marked as an obstacle). Some confusion may occur because flatw is, in practice, a two-dimensional array of unsigned chars in which the upper left corner is the address [0,0]. The ALV navigates in a coordinate system whose origin is always located at [row0, col0] in the current flatw (row0 and col0 are stored in the frame variable). Conversion from some (x,z) in range scanner world coordinates to an array address is done by:

$$\text{row} = \text{row0} - (z * z\text{ratio})$$

$$\text{column} = \text{col0} + (x * x\text{ratio})$$

As these equations suggest, $z$ratio tells the program how many pixels there are in the flatw array per unit of distance in the world along the z axis while $x$ratio gives the same information along the x axis. There is a separate ratio for the two axes to allow users to choose independently how coarsely they wish to model the world. $z$ratio and $x$ratio are fields in the frame variable.

As a side note, distances in IRS are often given in "range units". This term comes from the ERIM range scanner that is being modelled by IRS. In a range image produced by an ERIM scanner, one unit is equal to three inches. Of course, in the simulator, this correspondence to the real world is arbitrary.

4.3. Navigation and Path Planning Algorithms

IRS was primarily written to test low-level obstacle detection algorithms. It is anticipated that future researchers are likely to want to refine the higher level navigation algorithms . From the following description of the current process, it should be relatively simple to substitute improved algorithms in the appropriate functions.

The first time navigate() is called, it prompts the user to enter the location of the ultimate goal for the simulated ALV (which is stored in the variable "goal"). The goal is passed to find_path() where an initial subgoal is calculated. This subgoal is on the straight line from the ALV current's location to the ultimate goal. The distance along this straight line that the ALV will travel in a single move is determined by the compile-time constant Max_Move. Max_Move is defined in the file path.c. If the initial subgoal is located on a pixel in flatw that is not open (i.e. it's an obstacle or not in view), then the subgoal and flatw are passed to go_to_vertex() where the subgoal is moved to a nearby open pixel (the heuristics used by go_to_vertex() are described in the source code comments in path.c).

Once an open pixei is selected, the path planner described in Appendix B is applied to the subgoal. If the planner finds a path then find_path() terminates. Otherwise, cross_obstacle() generates a new subgoal that is designed to avoid the unreachable old subgoal. The user is warned that certain pathological patterns in the flatw map could lead to an infinite loop between the path planner and cross_obstacle().

The subgoal found by find_path() is kept in navigate() in the variable called "move". The function goal_reached() is called by navigate() to test whether "move" is within some small distance of the ultimate goal. If it is, a message is printed for the user and the program terminates. If not, then the next step is to calculate a transformation matrix that moves the ALV to "move". More exactly, a transformation matrix (named "omtm") is calculated that will translate the current coordinate system to a new one whose origin is located at "move". The matrix also rotates the coordinates so that the new z axis is pointed at the ultimate goal location. The function make_new_transform() calculates omtm. It also applies omtm to the variable "goal" so that the variable always contains the ultimate goal in terms of the current coordinate system. The function prints the values of omtm and the new goal. Once make_new_transform() is done, navigate() terminates and omtm is applied to each object's control points by c_process(). This is the beginning of the next pass of the simulator.

Note that omtm is the local parameter name for the global variable OBSV_MOTION_T_MAT. Each time c_process() is called, it prints OBSV_MOTION_T_MAT and CURR_OMTM_PROD. The latter matrix was used by IFS but now it simply has the same value as OBSV_MOTION_T_MAT so printing both of them is redundant. These matrices are in the IFS world coordinate system not the range scanner system.

4.4. Default Values in Images

Several images in IRS are initialized to a particular value that is used later in the program to indicate that a pixel has not yet been assigned a meaningful value. Most of these conventions are discussed in comments in the source code but they are collected here for convenience. In general, images that are global variables are assumed to be initialized to zero. Local images whose first dimension are pointers that are calloc'd or malloc'd are also assumed to be zero. These two assumptions are consistent with standard C conventions.

In refreshbuffer(), the global array "pic" has all of its entries set to the constant BLACK. Pic is the array that holds the gray level values of the current image. BLACK was defined to be 0 in prog.h so this is of interest only if a user wishes BLACK to have another value. Also in refreshbuffer(), the global array "$z$buffer", which holds the z value for obstacle pixels in pic, has every entry initialized to the constant INFINITY. This initialization is used later in two functions: 1) when a 3D point is being projected onto the image plane in colorin(), the point is assumed to be visible only if its z value is less than the $z$buffer value at the corresponding pixel (which is why $z$buffer must be initialized to a large value); and 2) when save_range_image() calculates a scene's range image using $z$buffer, the function knows that a range cannot be calculated wherever $z$buffer has a value of INFINITY.

An ERIM laser range scanner has a field of view that, when projected into a flat ground plane, is a trapezoid. IRS assumes that within this trapezoid, every pixel in the ground plane map is navigable unless it is explicitly identified as an obstacle or within the shadow of an obstacle. This assumption is implemented in init_values() where the ground map "flatw" is initialized to have a trapezoid of navigable pixels and all other pixels are marked as being out of range of the range scanner. The array "empty_flatw" is initialized with the same trapezoid pattern so that it can be used in subsequent calls of navigate() to re-initialize flatw without redoing the calculations done by init_values(). The source code comments in init_values() discuss the small difference between the first initialization of flatw and empty_flatw.

If more than one obstacle pixel in the range image maps into the same pixel in flatw, the program saves the tallest obstacle (because it will cast the largest shadow). The height (that is, the y coordinate) of an obstacle pixel is kept in the array "depth". Recall that the range-image coordinate system has its origin at the location of the range scanner and the positive y axis points down toward the ground so that the tallest obstacle is the one with the smallest value in the depth array. In the function make_flat(), all of depth's entries are initialized to the constant HUGE (HUGE is defined in math.h). This initialization ensures that obstacles mapped into flatw will always have a smaller value.

4.5. Creating the Visual Image

If the user desires more realistic visual images, it will be necessary to rewrite the functions that assign intensity levels to the array "pic". The process by which pic is assigned values begins whenever drawscene() is invoked. The global array "scene" holds each object that the user has created. For each object in scene, drawscene() calls the appropriate drawing function, i.e. drawcube(), drawcone(), drawsphere(), etc. Each of these drawing functions systematically sends groups of three control points to clip_and_color() until the entire surface of the object has been drawn. If any of the control points are behind the image plane, clip_and_color() calculates a new point so that the three points sent by clip_and_color() to colorin() are in front of the image plane. World_to_screen() is called at the start of colorin() to do two things: 1) project the three world coordinates into the image plane (actually, the image plane coordinates are not saved; instead, they are immediately converted into integer row and column values which are saved in the array "ip") and 2) calculate the gray level that will be assigned to all of the pixels within the triangle defined by the three projected points. The gray level is ultimately calculated in the function shade() by assuming Lambertian reflection at the center of the triangle without including the effect of diminishing brightness due to increased distance from the light source. This value is assigned to the "color" field in each of the three array points stored in ip. It would be relatively simple to calculate the intensity at each of the three points and interpolate those values in colorin() in the same way that the z value for each point in a projected triangle is interpolated from the z values of the three vertices in ip.

4.6. Miscellaneous Issues

When calculating the transformation matrix in make_transform(), the assumption is made that the ALV is driving to a flat location that will be at the same depth as the current location (i.e. y = scanner_height, where scanner_height is a constant in irs.h). If the simulated ground is not going to satisfy this assumption, the function will have to be modified.

When the ALV moves from one location to the next, the old ground map (which is kept in the array old_flat) is transformed in integrate() onto the current flatw. The old x and z coordinates are known from the location of the pixel in old_flatw. The y coordinate for obstacles is kept in the depth array. However, the depth array does not have the values for hidden pixels and open (i.e. navigable) pixels. Integrate() assumes that the y coordinate for these pixels is equal to scanner_height. If one wishes to remove this simplification, it will be necessary to modify make_flat() so that the function saves the depth of all pixels in "depth" instead of just calculating it for obstacle pixels.

The numerical stability of projections into the image plane is checked in two functions, colorin() and find_z(). Both functions are in the file colorin.c. The meaning of a warning is best understood by examining the source code and comments at the point in the file where the message is produced.

Once the "move" variable has been calculated, it is compared to the ultimate goal location, as described in Section 4.3. If the distance from "move" to the goal is small, the program stops without ever calculating the last transformation matrix that would actually drive the ALV to the "move" location. If the user wants the ALV to take this last step and produce the appropriate range and visual images, IRS will have to be modified in two places. First, in navigate(), delete the *else* in the code

```
if (goal_reached(goal, move))
    { *not_done_flag = FALSE;
     printf ("\n\n *** GOAL REACHED ****\n\n");
    }
else
    make_new_transform (move, omtm, &goal, inv_omtm);
```

so that make_new_transform() is always called. The function main() should be modified by adding a call to c_process() after the *while (not_done)* {...} loop.

APPENDIX A

Cross-Reference of Function Names

This is a complete listing of the functions in IRS, sorted alphabetically. The page and line numbers refer to source code listing printed 6/2/87. Due to peculiarities in the cross-referencer, some functions actually begin on the page after the one listed. All function names are truncated to 16 characters.

FUNCTION	FILE	PAGE	LINE
add_shadow	flat.c	94	332
affine	mat_trans.c	130	68
allocate_space	navigate.fun.c	164	23
angle_init	deriv.c	68	92
assign_curve_T_M	curve.c	53	152
c_algorithm	main.c	116	192
c_contour	c_contour.c	13	232
c_lightsource	c_contour.c	8	9
c_process	c_contour.c	14	253
c_scene	c_contour.c	9	28
clip	colorin.c	24	127
clip_and_color	newclip.c	173	39
colorin	colorin.c	27	160
cone	cone.c	37	3
copy_char_pic	navigate.fun.c	169	262
copy_float_pic	navigate.fun.c	170	281
copycylstruct	misc.c	155	168
copymat	mat_trans.c	140	230
copystruct	misc.c	154	158
copytobuff	Grinnell.c	104	40
copyvec	mat_trans.c	141	240
create_object_te	mat_trans.c	143	271
create_obsv_moti	mat_trans.c	142	247
create_surface_t	surface.c	217	71
createmat2	misc.c	151	48
createmat3	misc.c	149	6

cross_obstacle	path.c	187	235
cuboid	cube.c	42	26
cylinder	cyl.c	62	3
deriv_init	deriv.c	69	138
detect_obstacles	obst.c	177	16
develop	mat_trans.c	127	7
divisible	mat_trans.c	132	131
dotprod	colorin.c	26	142
draw2line	drawline.c	75	26
draw2line2	drawline.c	77	87
draw3line	drawline.c	74	6
drawaxes	misc.c	150	17
drawcone	cone.c	40	107
drawcube	cube.c	46	154
drawcurve	curve.c	60	379
drawcyl	cyl.c	65	104
drawfpts	curve.c	59	359
drawobjects	misc.c	152	100
drawscene	c_contour.c	17	376
drawsphere	sphere.c	212	120
drawsurf	surface.c	219	111
file_init_featpt	curve.c	51	81
find_z	colorin.c	34	492
find_end_point	newclip.c	175	115
find_grad	grad.c	99	129
find_minus	minus.c	146	109
find_path	path.c	183	23
find_theta	theta.c	223	130
frame_initialize	init_frame.c	110	31
funct	surface.c	216	64
getline	mat_trans.c	131	118
go_to_vertex	path.c	185	147
goal_reached	path.c	189	294
ground	path.c	190	323
help	main.c	118	235
helpcreate	c_contour.c	19	432
init_cone	cone.c	38	24
init_cube	cube.c	43	44
init_curve	curve.c	49	4
init_cyl	cyl.c	63	23
init_Grinnell	Grinnell.c	102	3
init_observer_mo	init.c	108	96
init_prev_featpt	curve.c	57	283
init_sphere	sphere.c	210	22
init_surface	surface.c	215	24

init_values	navigate.fun.c	165	104
initialize	init.c	106	5
integrate	map.c	121	20
interpolate	equi.c	83	219
is_in_view_cone	colorin.c	25	135
list_parameters	equi.c	85	262
main	main.c	112	5
make_binarymap	path.c	193	452
make_equiangular	equi.c	86	291
make_flat	flat.c	90	190
make_new_transfo	map.c	124	155
make_template	path.c	192	400
makeiden	mat_trans.c	133	143
matmult	mat_trans.c	138	202
navigate	navigate.c	157	20
open_cvl_read	read_cvl.c	198	63
or_images	obst.c	180	130
polaroid	mat_trans.c	129	47
print	mat_trans.c	128	26
print_char_image	navigate.c	161	204
print_float_to_c	navigate.c	163	289
print_int_to_cha	navigate.c	160	167
print_intptr_to_	navigate.c	159	127
print_parameters	equi.c	84	239
print_range_imag	saverange.c	202	92
printmat	mat_trans.c	139	216
putgraylevel	shade.c	207	68
putone	drawline.c	76	74
read_cvl	read_cvl.c	197	18
readreply	main.c	119	273
refreshbuffer	main.c	117	218
remove_ambiguity	navigate.fun.c	171	308
rotate_cone	cone.c	39	68
rotate_cube	cube.c	45	111
rotate_curve	curve.c	58	332
rotate_cyl	cyl.c	64	66
rotate_scene	c_contour.c	16	349
rotate_sphere	sphere.c	211	73
rotate_surf	surface.c	218	82
save_range_image	saverange.c	200	21
savescene	c_contour.c	18	401
shade	shade.c	204	3
sortony	colorin.c	23	77
sphere	sphere.c	209	3
start_cvl_read	navigate.c	162	239

surface	surface.c	214	4
swap	colorin.c	22	61
tball_init_featp	curve.c	55	212
theta_init	deriv.c	70	179
threshold	obst.c	178	58
toGrinnell	Grinnell.c	103	24
trackball_init	curve.c	52	130
trans	mat_trans.c	137	189
vecmag	shade.c	206	59
world_to_screen	colorin.c	21	13
write_cvl	write_cvl.c	226	14
write_cvl16	path.c	195	509
$x$rot	mat_trans.c	136	178
$y$rot	mat_trans.c	134	156
$z$rot	mat_trans.c	135	167

APPENDIX B

Path Planner Primer

These are directions for using the Puri/Kambhampati path planner program on the ALV Vax.

1. Make sure you have write permission for files /a/puri/qtrees/pathcoors and /a/puri/qtrees/pathlength. You will need execute permission for files in /a/puri/bin and /a/puri/qtrees. For some weird reason you also need to add the following file to your home directory, ~ /pro/umips/grinnell.l. The contents of this file should be copied from /a/puri/pro/umips/grinnell.l. You also need read permission for files in /a/puri/pro/umips.

2. Create a binary file in the following format. If your image has n rows and m columns then the file should contain n lines. On each line it will have m numbers. Each number will be 4095 or 4096. The numbers should be separated by a blank. $4095 = 0$ (= accessible pixel) and $4096 = 1$ (= obstacle pixel).

   The file should be in order from top of image to bottom and left to right (ie: raster scan order). For the sake of discussion, let's call this file "input_file". Since the image is going to be placed into a quadtree it must be square (ie: $m = n$) and n must be a power of 2.

   (IRS creates this type of a file and puts it into a file called "binary_map".)

3. The following pipeline transforms input_file into a quadtree suitable for use by a lisp path planning program. The quadtree output is kept in a file that I will call "_mapin". Note: _mapin MUST BE in the directory /a/puri/qtrees so do not choose a name that will trash an existing file of Puri's.

   ```
   /a/puri/bin/makpic width height < input_file | /a/puri/bin/r2q |
   /a/puri/bin/distransb width | /a/puri/qtrees/qset width >
   /a/puri/qtrees/_mapin
   ```

   Width and height are the number of columns and rows in the input image.

4. cd /a/puri/pro; mdlisp

5. You are now in maryland franz lisp. Type "(goto-fig 'path)".

This command loads many files and takes some time to perform.
Do not type the double quotation marks in the last sentence or in the following directions. They are only there to delimit the answers that you are supposed to enter. Do type the single quotation mark!
Now type "(setup-qtree-in-lisp)".
You will be asked for a filename, type "_mapin".

6. In a while the program will finish the last command and respond with the usual prompt "2_". Type "(trunc 35)", wait for the next prompt then type "(start)". The "(trunc #)" command tells the path planner to first find a coarse resolution path and then go back and resolve the details. The larger the #, the coarser the initial path. 25 or 35 are usually good values for this parameter (what's actually happening is that any node in the quadtree that has less than # nodes beneath it will be treated as a leaf node on the first pass of the planner planner).

7. The program will prompt you for the start point. This is the coordinate of the pixel in the image where the path will begin. The coordinate system has its origin at $(x,y) = (0,0)$ in the lower-left corner of the image. Starting from the origin, x = column and y = row. The next prompt will be for the goal point. This should be answered similarly to the start point prompt.

8. The program then plans a path and places a list of the path's pixels in the file "/a/puri/qtrees/pathcoors". If you are planning multiple paths you must save the contents of ".../pathcoors" before running the program a second time. The listing is actually the path in reverse since it starts with the node just before the goal node and ends with the node that comes just after the start node.

9. The program at this point will also prompt you for a filename to store path information in. You must give it a name. Let's call this "_file2". "_file2" will be placed in the directory /a/puri/qtrees so DO NOT CHOOSE A NAME THAT WILL TRASH AN EXISTING FILE OF PURI'S!
"_file2" contains data that you do not need unless you wish to print the path on the imagen. How one actually does this is beyond the scope of this direction sheet (i.e.: I don't know how to do it yet) but I think you can type "cprintpic" from the appropriate directory of Puri's and follow the prompts from there. Good luck.

10. You can now leave lisp by typing "bye". Or, you can suspend lisp by the usual "control-z". If you suspend lisp then the next time you start it up again you should not type "(goto_fig 'path)". This will save a little time. Warning: using control-z may or

may not work. It is not a fully explored option.

11. While you are in mdlisp, if you make a mistake and wind up in error mode (indicated by a prompt that looks like "#<#>") type the control key and the letter "d" simultaneously to return to run mode (indicated by the prompt "#_").

APPENDIX C

Thresholding Range Images

The range derivative algorithms used by IRS to detect obstacles depend on the user to set effective threshold levels. After the range derivatives have been found, detect_obstacles() calls threshold() to create a binary image. The first time threshold() is called, it asks if the user wants to use the automatic cutoff values for *all* thresholding (see line 593 of the transcript in Section 3). If the user answers "yes", then the threshold levels set in detect_obstacles() will *always* be used in threshold(). If the user answers "no", then each time an image is ready to be thresholded, the program will ask the user if the automatic cutoff level should be used for *that particular* image. If the user answers "no" for that image, then the program places the image in a file and lets the user threshold it themselves. The following transcript gives examples of these options. The numbered lines are from the actual transcript. Comments about the transcript begin with "==>". The user responses are underlined.

```
593    Shall all thresholding be done using automatic cutoffs?
594    Your reply must be either yes or no
595    no
596
597    Use automatic threshold level(= 3) for theta derivatives?
598    no
==>         The number in parenthesis is the threshold level that will be
       used if the user replies affirmatively.
599
600    You must suspend the program and threshold image in <temp.thresh>
file
601    Type "yes" when thresholding is finished
602    ^z
```

```
==>     If the operating system does not allow you to suspend a program
        by typing <control-z> or some other signal, then IRS will
        have to be modified to permit this interruption.
Stopped
 2 C: man.thresh temp.thresh temp.thresh
==>     "man.thresh" is a local thresholding program. Any thresholding
        program could be used but the resultant binary image must
        be placed in the file "temp.thresh" before IRS is restarted.
Give integer threshold level:
7
 3 C: fg
==>     Restarting IRS.
IRS irs.parms d d
yes

Use automatic threshold level(= 4) for phi derivatives?
yes

Use automatic threshold level(= 3) for minus derivatives?
no

You must suspend the program and threshold image in <temp.thresh> file
Type "yes" when thresholding is finished
^z
Stopped
 4 C: thresh temp.threshtemp.thresh
Give integer threshold level:
12
 5 C: fg
IRS irs.parms d d
yes

Do you wish to save obstacle array?  yes
```

If the user knows a priori what the correct thresholding cutoffs will be, then the automatic values in detect_obstacles() can be set to them. Currently these values are compile-time constants but it would be trivial to alter detect_obstacles() or threshold() to allow the cutoffs to be entered at run-time.

REFERENCES

D. Dementhon, "Production of Smooth Range Images from a Plane-of-Light Scanner", Center for Automation Research Technical Report, College Park, Maryland, 1987. (To be published.)

S. Kambhampati and L.S. Davis, "Multiresolution Path Planning for Mobile Robots", *IEEE Journal of Robotics and Automation* **RA-2** (1986), 135–145.

S. Puri and L.S. Davis, "Two Dimensional Path Planning with Obstacles and Shadows", Center for Automation Research Technical Report 255, College Park, Maryland, January 1987.

S.S. Sinha and A.M. Waxman, "An Image Flow Simulator", Center for Automation Research Technical Report 71, College Park, Maryland, July 1984.

P.A. Veatch and L.S. Davis, "Range Imagery Algorithms for the Detection of Obstacles by Autonomous Vehicles", Center for Automation Research Technical Report 309, College Park, Maryland, July 1987.

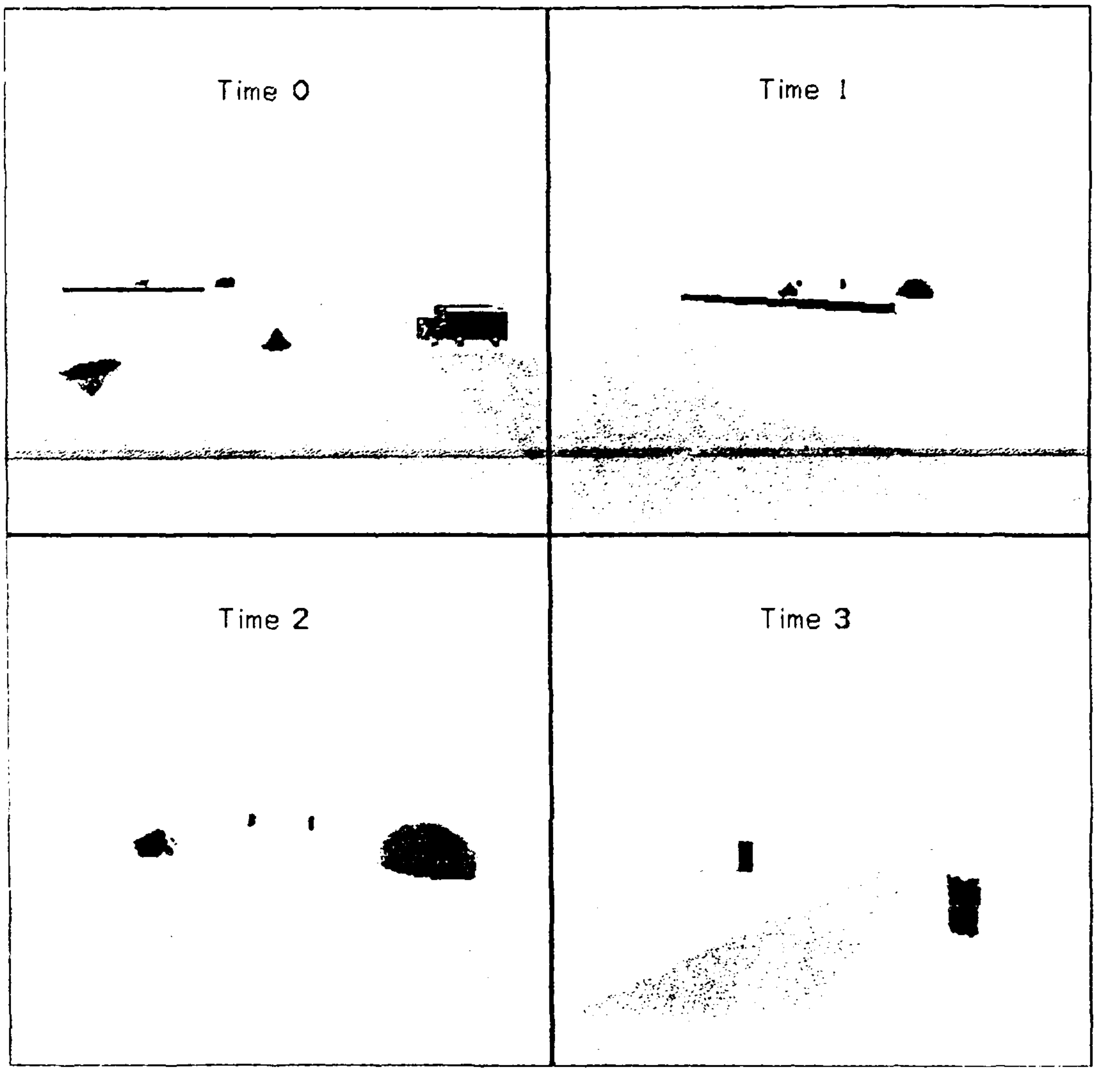


Figure 1: Visual Images from ALV Simulator

Figure 2: Original Range Image from Time 0 of Simulator

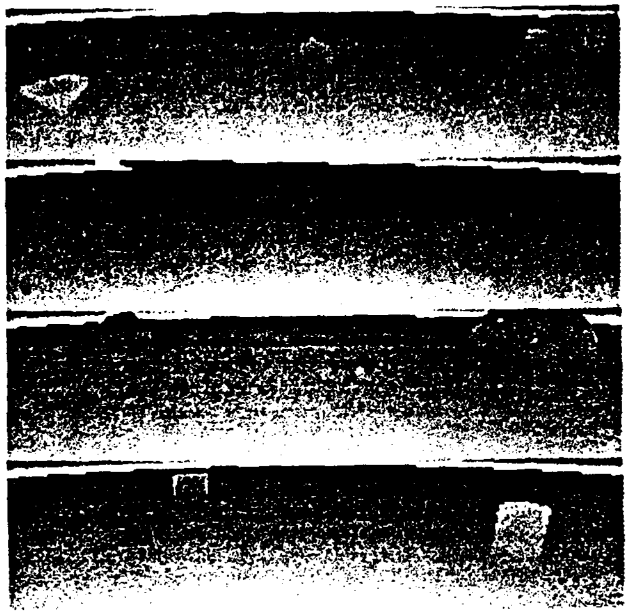


Figure 3: Montage of Original Range Images from ALV Simulator

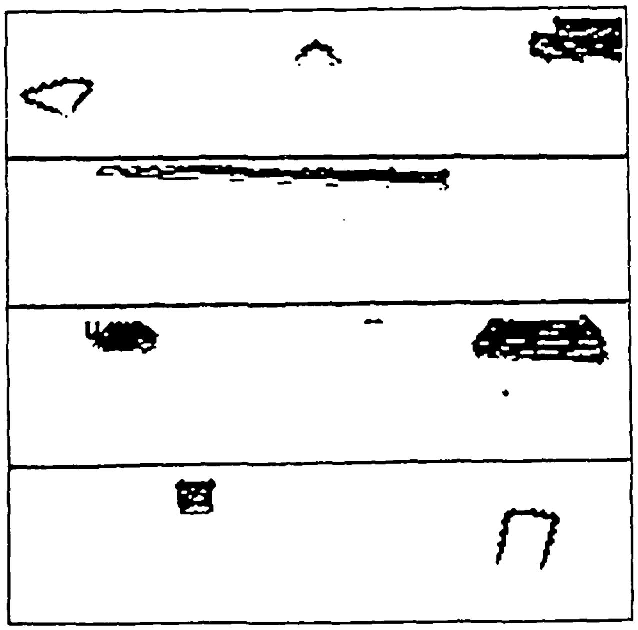


Figure 4: Montage of Thresholded Obstacles

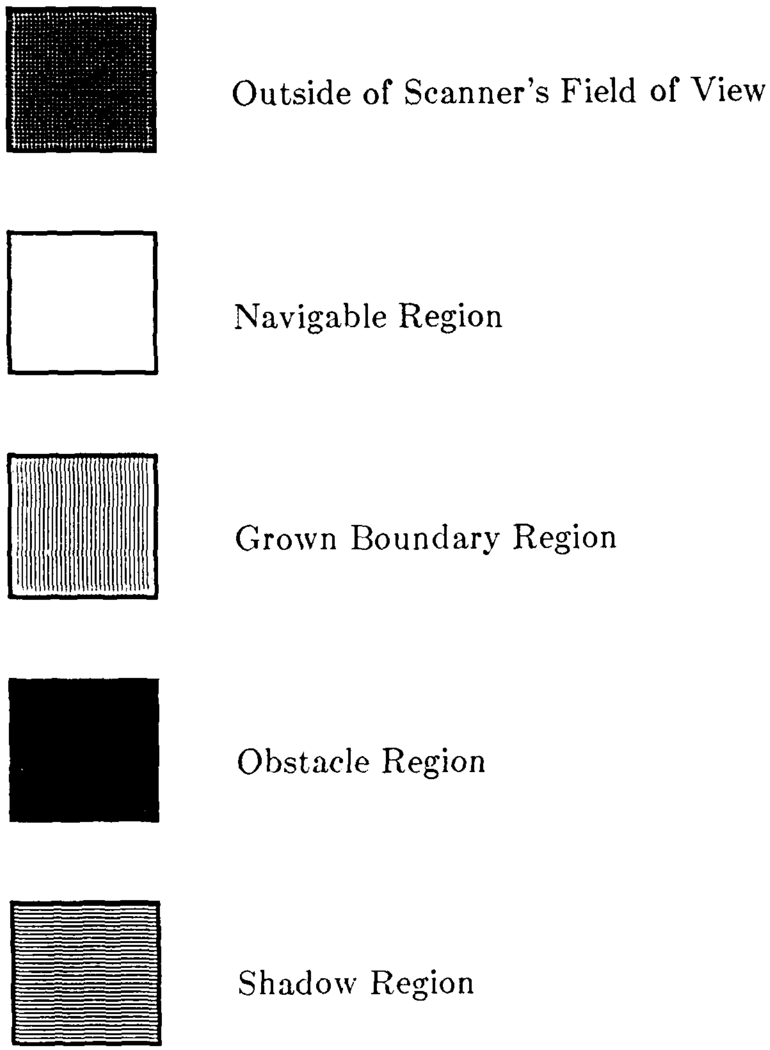


Figure 5: Key for Ground Plane Maps

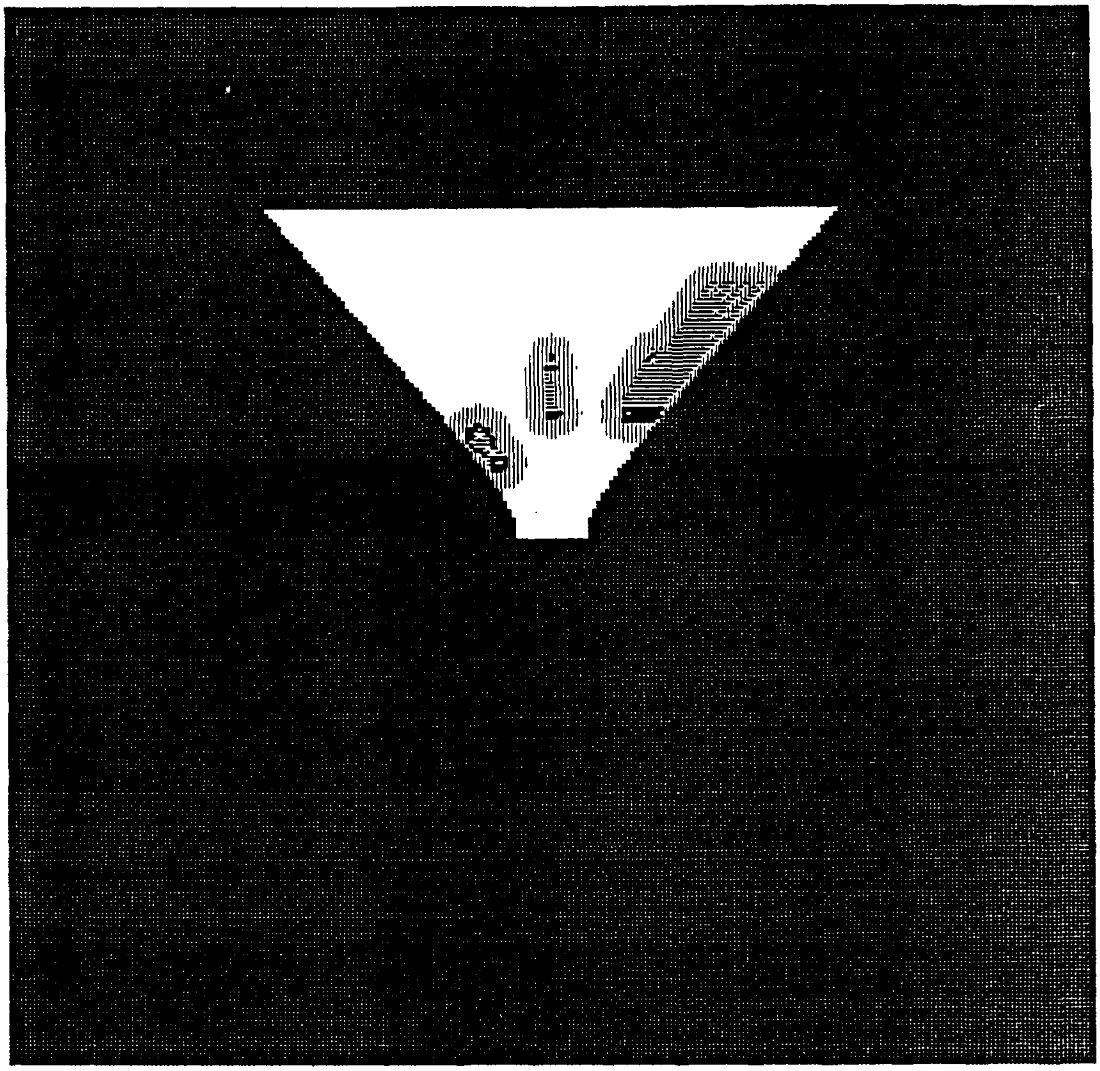


Figure 6: Ground Plane Map from Time 0

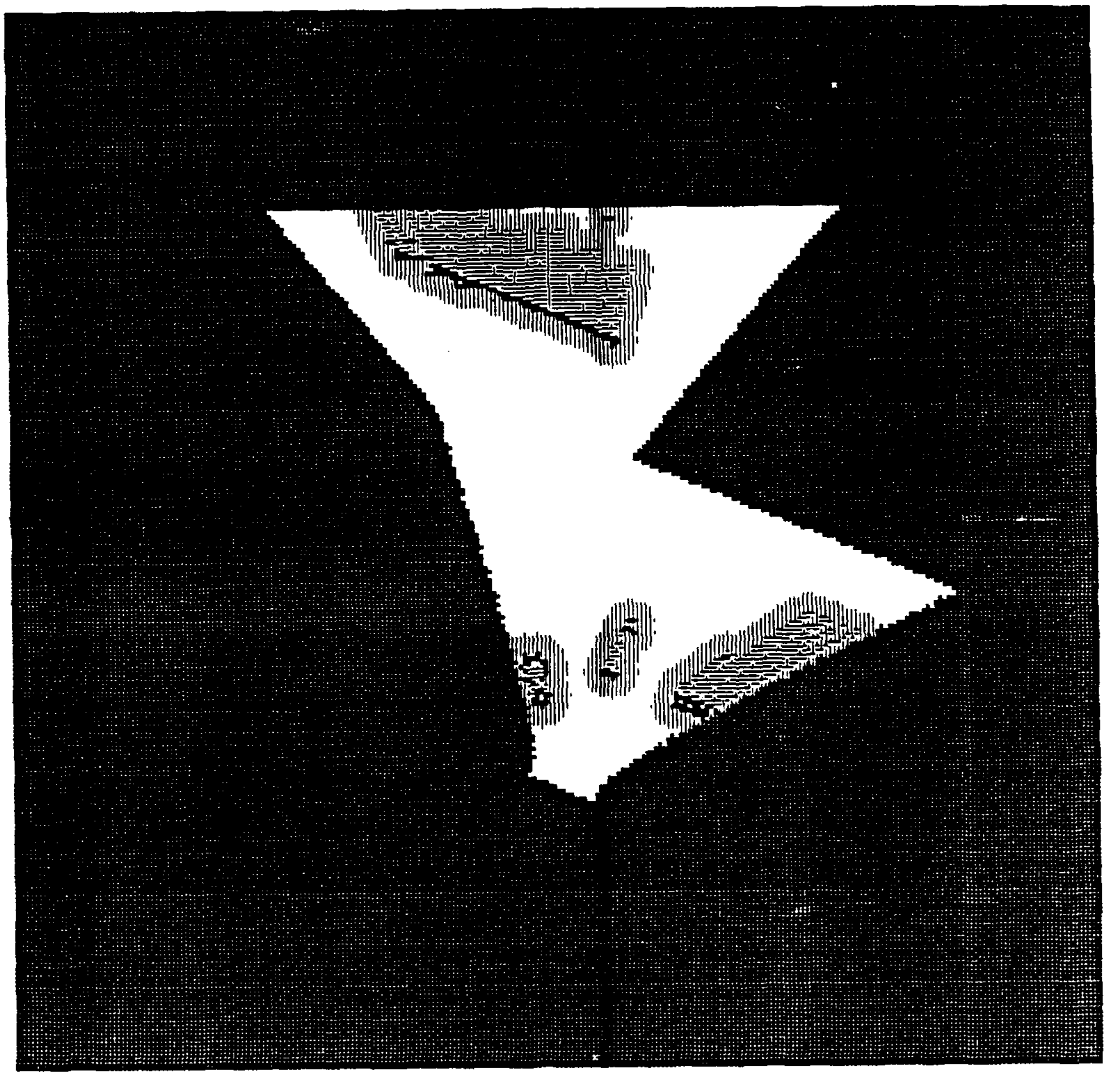


Figure 7: Ground Plane Map from Time 1

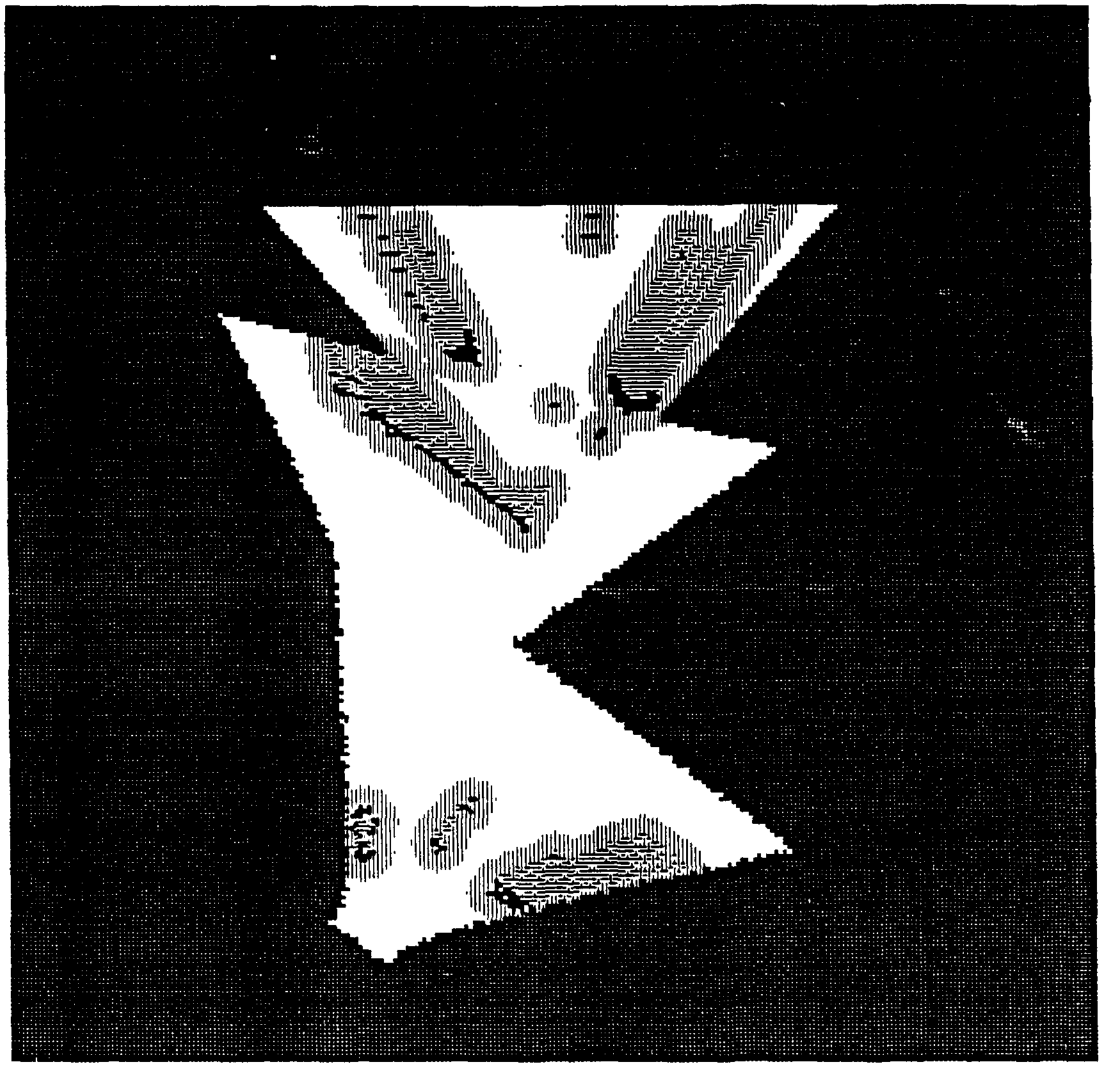


Figure 8: Ground Plane Map from Time 2

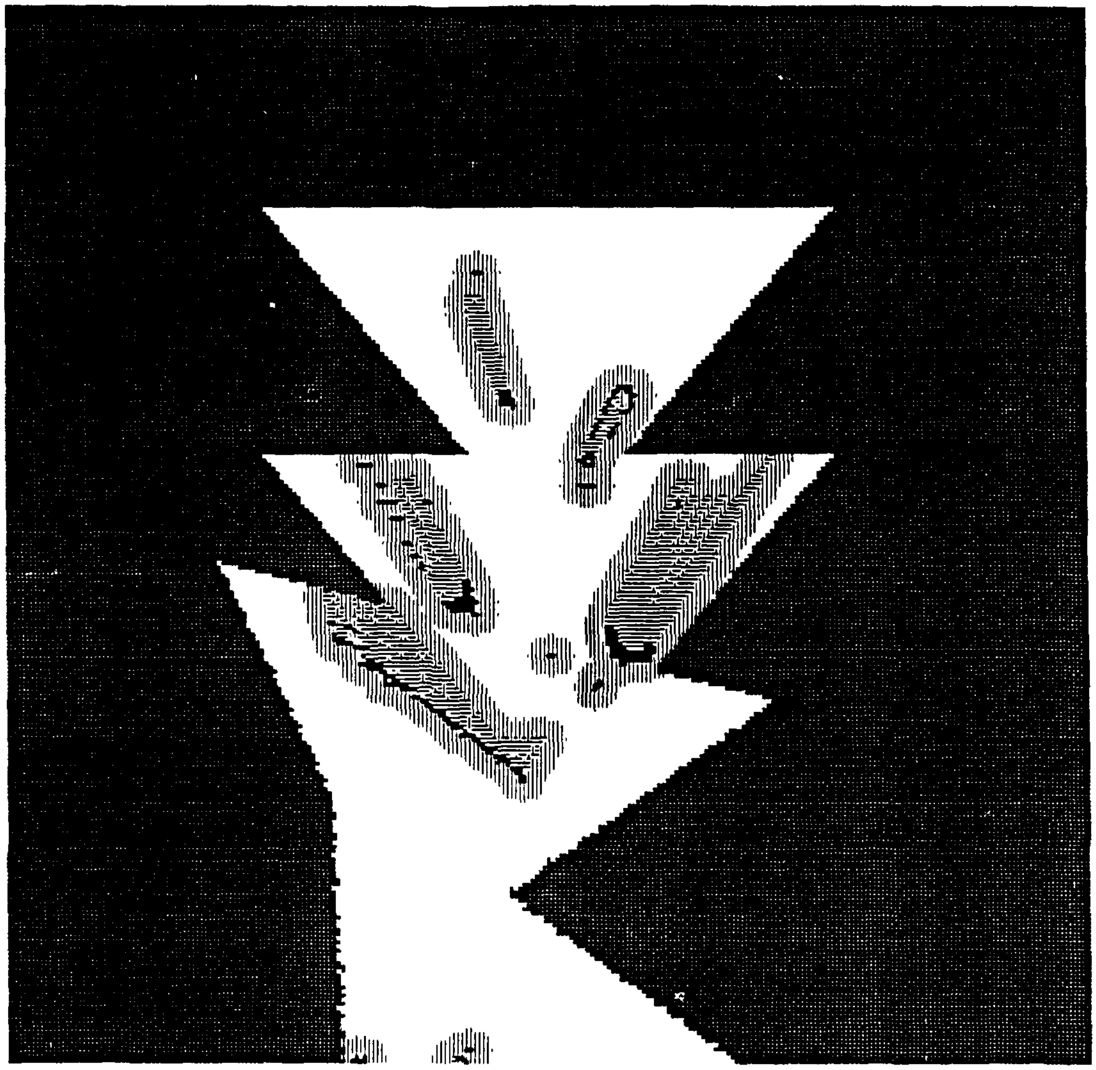


Figure 9: Ground Plane Map from Time 3

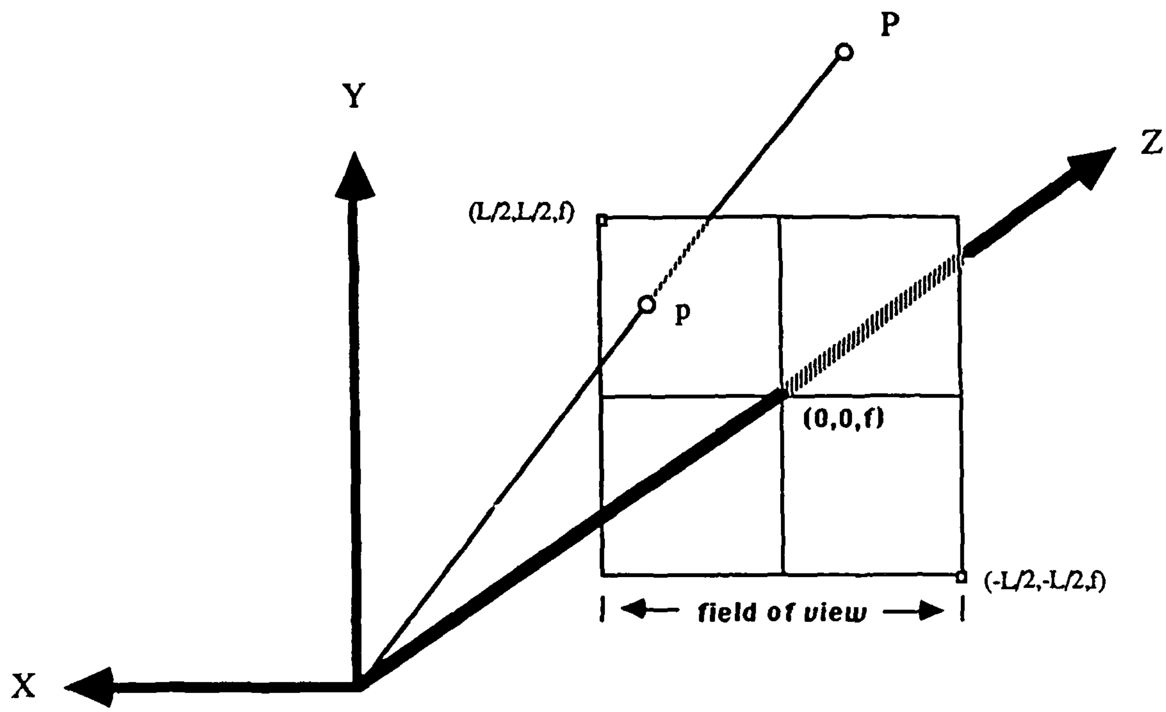
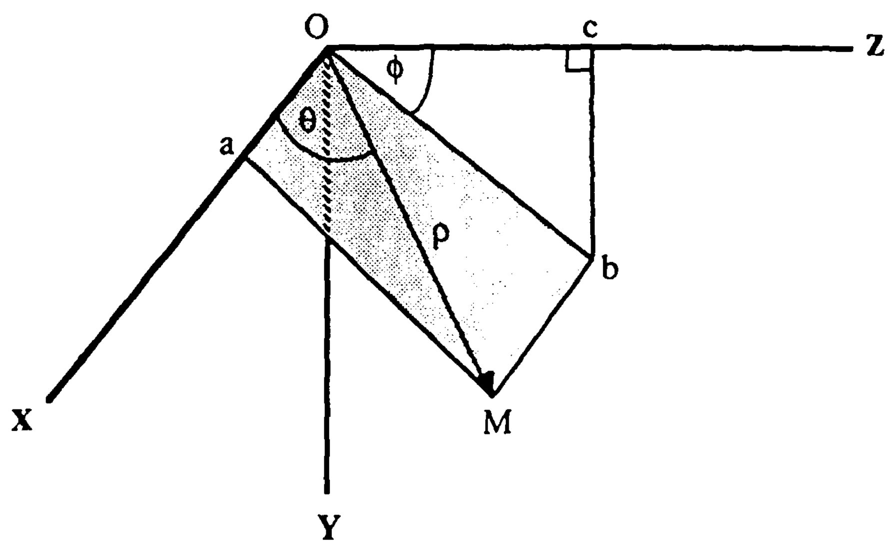


Figure 10: IRS World Coordinate System



ρ = range

ϕ = vertical scan angle

θ = horizontal scan angle

Figure 11: Range Image Coordinate System