NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

# RANGE IMAGE PROCESSING FOR LOCAL NAVIGATION OF AN AUTONOMOUS LAND VEHICLE

by

Dennis Duane Poulos

September 1986

Thesis Advisor: R. B. McGhee

CURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| SECURITY CLASSIFICATION AUTHORITY<br><br>DECLASSIFICATION / DOWNGRADING SCHEDULE | 3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release, distribution is unlimited |
| PERFORMING ORGANIZATION REPORT NUMBER(S) | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| NAME OF PERFORMING ORGANIZATION<br>Naval Postgraduate School<br>6b OFFICE SYMBOL (If applicable)<br>32 | 7a. NAME OF MONITORING ORGANIZATION<br>Naval Postgraduate School |
| ADDRESS (City, State, and ZIP Code)<br>Monterey, California 93943-5000 | 7b. ADDRESS (City, State, and ZIP Code)<br>Monterey, California 93943-5000 |
| NAME OF FUNDING / SPONSORING ORGANIZATION<br>USACDEC<br>8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>MIPR ATEC 46-86 |
| ADDRESS (City, State, and ZIP Code)<br>Ft. Ord, CA 93941 | 10 SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO |

TITLE (Include Security Classification)
RANGE IMAGE PROCESSING FOR LOCAL NAVIGATION OF AN AUTONOMOUS LAND VEHICLE

PERSONAL AUTHOR(S)
Dennis Duane Poulos

| a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT (Year, Month, Day) | 15 PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM ______ TO ______ | 1986 September 26 | 206 |

SUPPLEMENTARY NOTATION

| COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Artificial Intelligence; Image Processing; Robotics |
| | | | |
| | | | |

ABSTRACT (Continue on reverse if necessary and identify by block number)
A central emphasis for robotic research over the last few decades has been to make the systems autonomous. This implies simulating the human senses with electronic sensors and then deriving knowledge about the environment from those sensors. A large base of research exists that concentrates on computer vision algorithms that attempt to duplicate human vision. For the most part, this research has concentrated primarily on two dimensional techniques due to limitations in the available optical technology.

Real time range image processing is now feasible with the introduction of the ERIM Laser Scanner as installed on the Ohio State Adaptive Suspension Vehicle (ASV). The purpose of this thesis is to utilize the three dimensional data from the Laser Scanner and by rule-based programming techniques generate a local terrain feature model for use by an Autonomous Land Vehicle. Future applications of this technology include an autonomous Lunar Rover or autonomous service/repair robots operating in close proximity to the Space Station.

| | |
|---|---|
| DISTRIBUTION / AVAILABILITY OF ABSTRACT<br>[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS | 21 ABSTRACT SECURITY CLASSIFICATION<br>UNCLASSIFIED |
| a NAME OF RESPONSIBLE INDIVIDUAL<br>Prof R. B. McGhee | 22b TELEPHONE (Include Area Code)<br>(408)646-2552 / 22c OFFICE SYMBOL<br>52Mz |

) FORM 1473, 84 MAR    83 APR edition may be used until exhausted
All other editions are obsolete    SECURITY CLASSIFICATION OF THIS PAGE

# Range Image Processing For Local Navigation Of An Autonomous Land Vehicle

by

**Dennis Duane Poulos**
Lieutenant Commander, United States Navy
B. S., United States Naval Academy, 1975

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

NAVAL POSTGRADUATE SCHOOL

September 1986

## ABSTRACT

A central emphasis for robotic research over the last few decades has been to make the systems autonomous. This implies simulating the human senses with electronic sensors and then deriving knowledge about the environment from those sensors. A large base of research exists that concentrates on computer vision algorithms that attempt to duplicate human vision. For the most part, this research has concentrated primarily on two dimensional techniques due to limitations in the available optical technology.

Real time range image processing is now feasible with the introduction of the ERIM Laser Scanner as installed on the Ohio State Adaptive Suspension Vehicle (ASV). The purpose of this thesis is to utilize the three dimensional data from the Laser Scanner and by rule-based programming techniques generate a local terrain feature model for use by an Autonomous Land Vehicle. Future applications of this technology include an autonomous Lunar Rover or autonomous service/repair robots operating in close proximity to the Space Station.

## TABLE OF CONTENTS

## ACKNOWLEDGMENTS

I would like to take this opportunity to thank Captain Robert Richbourg for his assistance. He spent a number of hours helping me with the "C" program to dump the ISI bitmaps and answering a number of questions I had about graphics in general. His help, in particular, saved many long hours of tedious work.

Special recognition is due to Professor Abbott. Due to unforeseen circumstances, I found it necessary to impose on him at the last minute and ask him to act as second reader for the thesis. He kindly agreed, devoting a large amount of time from his already busy schedule to do a very professional job. His comments added significantly to the accuracy and readability of the presentation.

I would especially like to acknowledge the assistance and encouragement that my advisor, Professor Robert McGhee, gave me throughout this research. He accepted me as a thesis student late in the program and at a time when he already had a large number of other students to advise. His guidance and advice were invaluable and I am extremely grateful to him for his time and counseling.

Lastly, I would like to thank my parents and my family for their support. It is difficult under the best of circumstances to complete a Masters program but with three children and a family to support it is impossible without a very understanding, helpful, and loving wife. Thank you.

# I. INTRODUCTION

## A. GENERAL BACKGROUND

Space is an extremely hostile environment. Whether in free space, orbiting the earth, or on the surface of foreign worlds. mankind is ill equipped to work or even survive for long periods of time. Yet mankind is driven by its natural curiosity to explore this vast frontier. Not only is mankind attempting to explore space but plans are underway to build habitats in space and to live off earth in the near future. There are bold concepts being developed for factories in space, huge colonies of people orbiting between the Moon and Earth, and manned landings on Mars within the next fifty years. However, the weak link in all these ambitious plans is mankind itself. Tied to bulky space suits and physiologically limited by the amount of exposure to microgravity and radiation that can be withstood, mankind is barely capable of doing the large scale manual labor required to bring these dreams to fruition. Consequently, there are those that feel that if mankind is to ever truly become a spacefaring being it will have to be done through a symbiotic relationship with intelligent machines or *robots.*

In this relationship. robots will be utilized to perform those functions for which it is not economically feasible to task humans with. These functions will include manual labor such as the construction and maintenance of large scale space structures where transportation costs will make it prohibitive to employ people as common metal workers. Robots will also be delegated the tasks too boring or dangerous for mankind to perform such as doing long term exploration missions on the surface of the planets which mankind may wish to investigate. Certainly, mankind will soon return to the surface of the Moon and will someday land on Mars but it will be robots which will be tasked to spend the boring weeks and months exploring the surface.

For its part. mankind will perform those tasks which robots are ill equipped to do. These include analyzing the data and samples returned by robotic explorers, doing precision manual labor such as maintaining electrical equipment, and other tasks requiring a high degree of cognitive or reasoning capability. Mankind will be the thinking part of the team and the robots will be the muscle.

But just as mankind will. on occasion, be required to use muscle in the conquest of space, so will robots be required to exhibit intelligent behavior. An autonomous vehicle exploring the surface of Mars would be an economic disaster, considering transportation and fuel costs, if every time the autonomous vehicle came to a cliff it would just walk right over the edge or if the vehicle ran into every hazard in its path instead of taking the required action to avoid that hazard. However, today the technology required to produce the necessary intelligent behavior in robots is just beginning to be developed. These technologies include artificial intelligence programming, walking technology, and vision sensors to name but a few.

The purpose of this thesis will be to investigate, by simulation, a new algorithm for computer vision for autonomous vehicle navigation and path planning. This algorithm will attempt to classify terrain by type rather than specifically identify individual terrain items.

## B. ORGANIZATION

A discussion of the recent history in computer vision research is presented in Chapter 2. This chapter includes presentations on some of the basic work in vision research, a number of the different sensor types used to implement vision, and the most recent work utilizing state of the art testbeds. Finally, the background for the terrain classification algorithm implemented in this simulation is addressed.

In Chapter 3, a detailed mathematical development of the formulas used in the simulation is given. A complete understanding of the mathematics is required to understand the limitations on the algorithm and some of the requirements necessary in implementing it in hardware.

Chapter 4 provides a description of the program written for the simulation. This includes a brief discussion of the LISP progamming language and bitmap graphics, both of which are important to this simulation. Finally, the methodology for displaying the symbolic information generated by the program is discussed.

The symbolic results from the simulation are presented in Chapter 5. The simulation produces images of the terrain scenes analyzed and symbolic maps of each scene as generated by the algorithm. Each scene is analyzed under a number of different conditions. These conditions include both noiseless as well as noisy data and varying internal thresholds. The effect of each of these variables is briefly discussed.

Finally, Chapter 6 summarizes the contributions made by this research and discusses some of the possible areas of further research based on this work. Three appendices, a secondary mathematical development of the formulas as based on matrix algebra, the graphics routines utilized, and the simulation as it is written in LISP, are included after Chapter 6.

# II. SURVEY OF PREVIOUS WORK

## A. INTRODUCTION

To make a land vehicle truly autonomous, the principles of robotics have to be applied in its design. These principles include giving the vehicle the ability to perceptive and understand the local environment from sensor data and the artificial intelligence to plan it's actions as a result of this understanding. To one degree or another, all animal life shares a common set of environmental sensors and so. intuitively, those sensors are the ones that roboticists attempt to incorporate in their designs.

The set of natural senses includes touch, taste, hearing, smell, and sight. Taste and smell are primarily chemically-based sensors. Touch and hearing are pressure related phenomenon. Finally, the ability to sense temperature remotely and to see are essentially passive electromagnetic capabilities that operate in two different wavelength regimes. Although all the senses must be fully operational for a person to feel comfortable in their ability to completely sense the local environment, the one natural sense that seems to be the primary sensor, at least for human beings, is vision.

While all the senses are normally required, a person is able to adequately perform basic functions even with the loss of most of the other senses so long as the vision system is not impaired. Accordingly, the ability to simulate vision is one of the essential requirements for a high performance autonomous vehicle [Ref. 1:pp. 79-121]. This ability to simulate vision has been, over the last few decades, one of the central areas of robotics research.

Until recently, most vision research has had to contend with extracting information from two-dimensional data only. This is because the conventional mechanism for passively receiving electromagnetic energy has been a camera of some sort. Whether these cameras have been standard TV cameras, CCD devices, or sensitive IR receivers, their output is two-dimensional in nature. To most researchers, this is acceptable since that is the way humans see.. However, it now appears that the human vision mechanism is extremely sophisticated in the way that it processes this two-dimensional data to extract the third dimension of range from a scene. The massive parallelism used by the brain in its signal processing algorithms will be very difficult to duplicate, at least in the near future [Ref. 2:pp. 180-190]. Clearly, though, range is an important type of information for any autonomous vehicle. Consequently, work in robotic image processing has progressed from two-dimensional image segmentation and pattern recognition into active three-dimensional image sensing. [Ref. 3:pp. 206-220 ; Ref. 4:pp. 3-12 ; Ref. 5:pp. 1-15]

The purpose of this chapter is to introduce some of the specifics of range sensing, the techniques of scene description, and finally, to examine previous approaches to feature extraction based on three-dimensional imaging.

## B. APPROACHES TO ACQUIRING RANGE IMAGES

### 1. Conventional Radar

The earliest technology for directly acquiring a range image was conventional radar. The term RADAR is an acronym for RAdio Detection And Ranging. As the acronym implies, radar emits radio frequency energy and uses the information contained in the reflected beam to derive information regarding the target. The beam of radio frequency energy can be in pulse form or continuous wave form [Ref. 6:pp. 1-40,359-391].

For conventional radar, the usual format is the pulse form in which the radar beam is pulsed on and off at a specified rate. This rate is called the *pulse repetition frequency* , PRF. The length of time that the pulse is on is called the *pulse width* , PW. These two parameters, along with the operating frequency and aperture dimensions of the transmitting/receiving antenna, define most of the important characteristics of the radar.

#### a. Range Measurement

The range information in a radar system is derived from time of flight calculations. The pulsed radar beam travels at the speed of light, C, $3 \times 10^8$ meters per second. By measuring the time from when the beam is transmitted to when the reflected beam returns, and dividing that time by two to account for the two way trip, the range can be calculated as:

$$R = \frac{\Delta(t)}{2} \times C. \tag{2.1}$$

If this was all there was to measuring range, then the range to any target could be determined exactly.

Unfortunately, this is not the case. Receiving mechanisms have a limited threshold for detecting the reflected signal. A minimum amount of energy has be be received before the detector can determine that the energy is a signal and not just noise. The amount of time after a receiver starts receiving a reflected signal until it can classify the signal as more than noise is a function of the power in the returned signal, since energy is defined as power times time. The time lost in signal determination introduces a error in the range calculation. As the range to the target increases, the power in the received signal is reduced as a function of the fourth power of the range, so the time from the instant when the signal is initially received till when it can be classified as true signal increases as a function of range. This variation in power with distance contributes another significant error in the range determination. Finally, few radars are capable of actually determining the range from the leading edge of the pulse only. Instead, information is derived from the entire length of the pulse. Therefore, the pulse width is commonly used as a lower limit for the range resolution of the radar beam. For example, a PW of .3 microseconds has an inherent range resolution of 90 meters. This distance is also considered as the lower limit on the minimum range that a radar can measure.

An obvious solution to the problem of range resolution would be to reduce the PW to an absolute minimum. To a degree. this is in fact done. However. minimizing PW is not without its own set of limiting problems. The initial power of the transmitted pulse has to be large enough so that the energy in the returned pulse will exceed the thresholding limits of the receiver. So, as pulse width is reduced to increase range resolution, the power of the RF pulse generator has to go up proportionately.

The PRF also has a significant effect on range measurement with conventional radar. Generally, the antenna that is used to transmit the high power radar beam is the same one that is used to receive the extremely weak reflected signal. Clearly. the radar cannot transmit and receive at the same time. If there is even a small impedance mismatch at the antenna/waveguide interface. then a portion of the radar signal is reflected back to the transmitter/receiver. If the receiver were on while the beam was being transmitted, the energy reflected by the interface would, at a minimum, swamp out the returning signal and. in the worst case, it could physically destroy the extremely sensitive receiver.

The maximum range the radar can detect is defined by the PRF. The on and off time of the transmitter defines one cycle. The PRF is the number of on-off cycles per second. The on time during one cycle is defined by the PW, so that:

$$off-time = \frac{1}{PRF} - PW \; . \tag{2.2}$$

During the off time. the beam has to travel to the target and back again to the receiver. The maximum detectable range is. then. one half the pulse off-time times the speed of light.

The radar parameters that directly effect the ability to measure range are. in summary. pulse width. which determines power and range resolution, and pulse repetition, rate which determines maximum range.

b. Lateral Measurement

Raw radar spatial measurements are usually recorded in spherical coordinates. Spherical coordinates are defined by two angles. usually theta ( $\theta$ ) and phi ( $\phi$ ). and range. r. Theta is an azimuth angle measured either between -$\pi$ and $\pi$ or 0 and $2\pi$ radians from a known reference. Phi is an elevation angle measured between -$\pi/2$ and $\pi/2$ from a reference elevation (usually the horizon). Range and the ability to resolve range has been discussed in the previous paragraphs. Measuring elevation and azimuth are relatively simple in radar. The elevation and azimuth are simply the measured position of the antenna at the time of transmit/receive. However. resolving differences between small angular displacements places a limit on the ability to measure physical dimensions and angular resolution is a function of the beam width.

From optical diffraction theory. a generalized formula for determining the beam width. and therefore the angular limit of resolution, of an electromagnetic beam can derived [Ref. 7:pp. 353-352 : Ref. 8:pp 451-464]. This angular limit is defined as the radius of the Airy Disk of the beam focused at infinity and is given by:

$$\Delta\theta = 1.22\lambda/D \tag{2.3}$$

where $\lambda$ is the wavelength of the radar and D is the diameter of the antenna (or focusing lens). This result is in radians. To get an answer in terms of resolvable distance, $\Delta\theta$ is multiplied by the range to the target; i.e.,

$$d = \Delta\theta \times Range \tag{2.4}$$

As an example, the wavelength for a 1 ghz radar beam is 10 cm. Assuming a 1 meter antenna and a range to the target of 1000 meters, the minimum resolvable separation will be 122 meters. This is acceptable for target detection and a rough determination of azimuth, but for an autonomous vehicle, this is not very useful.

c. Continuous Wave Radar

Continuous wave radar is not discussed as an imaging technology since it has limited capability as a imaging system. It is generally used as a target illumination system or as a doppler moving target detector. Range detection is limited to continuous wave radars which have been modulated by a periodic wave and the range accuracy of such systems is limited [Ref. 6:.pp 359-390].

#### d. Conclusions

Radar was one of the earliest technologies to use some of the concepts of range imaging. The limitations of radar, though, which are inherent in the required pulse widths, pulse repetition frequencies, power limitations, and beam width due to frequency, make it an unattractive solution to the range imaging requirements of a terrestrial autonomous vehicle.

### 2. Infrared, Millimeter Wave and SAR

Higher frequency (shorter wavelength) active systems and advanced technology systems offer increased accuracy in pointing and range measurement. These systems include infrared, millimeter wave technology and synthetic aperture radars. At present, there is significant research into this kind of advanced technology. However, for now, their capabilities are limited in scope when used to analyze natural scenes. Also, the equipment is generally either large, fragile, unreliable, or expensive. Accordingly, the military is the main user of these advanced technologies and their adaptation to civilian uses (such as robotics) will take time. [Ref. 1:pp. 107]

### 3. Optical Stereo

Another region of the electromagnetic spectrum in which the receiver technology is rather well developed, though less sensitive, is the visible region. A common way to generate range information in this regime is by the use of a technique called *disparity stereo* or *binocular vision.* As shown in Figure 1, this technique requires two optical sensors in the same plane, viewing the same scene

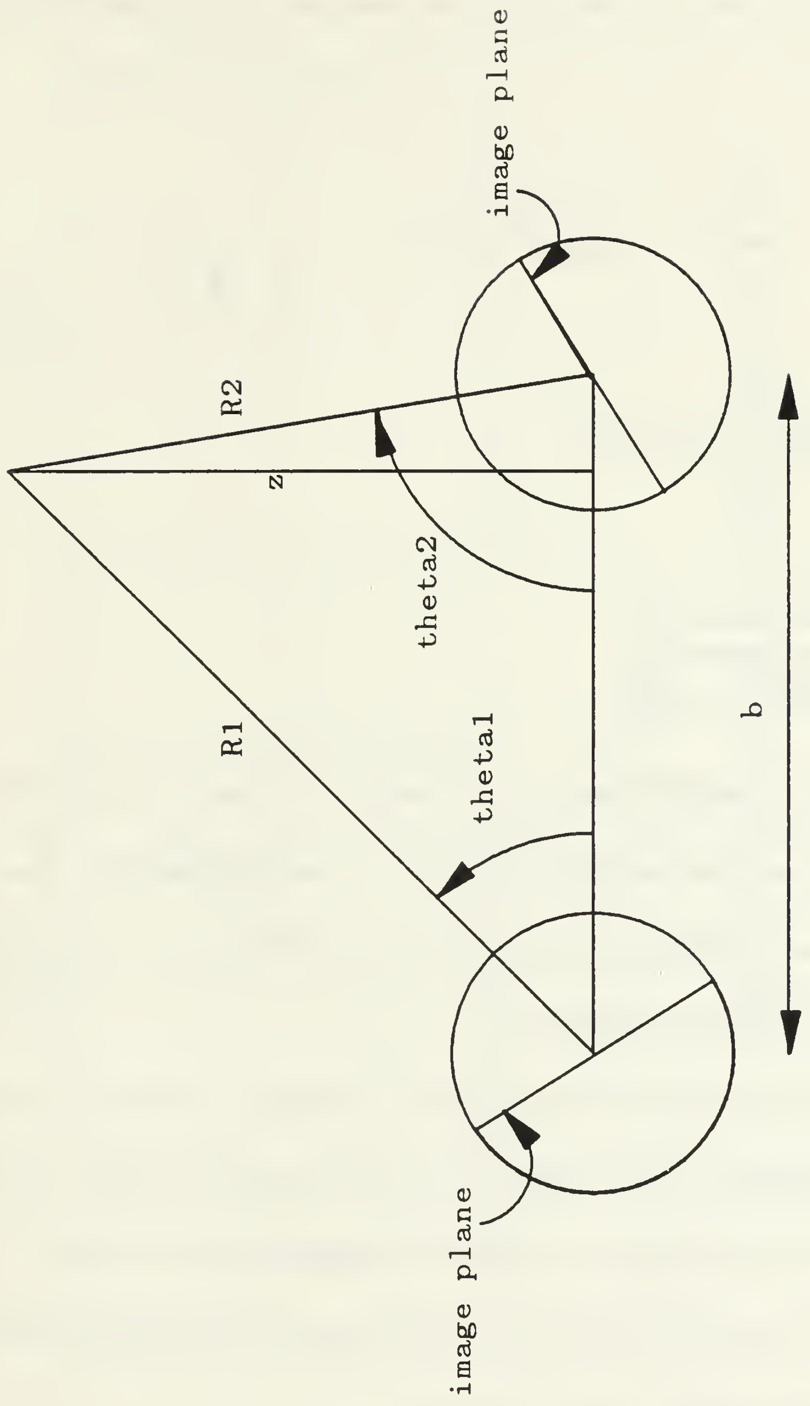

Figure 1. Binocular Vision

with their central axes parallel. Range. z. to the target. P. is measured in terms of the angles $\theta_1$ . $\theta_2$ and the baseline. b. between the two receivers. From the law of sines

$$\frac{R_1}{sin(\theta_2)} = \frac{R_2}{sin(\theta_1)} = \frac{b}{sin(180^0-\theta_1-\theta_2)} \tag{2.5}$$

but

$$z = R_1 sin(\theta_1) = R_2 sin(\theta_2) \tag{2.6}$$

so

$$z = \frac{b\ sin(\theta_1) sin(\theta_2)}{sin(180^0-\theta_1-\theta_2)} . \tag{2.7}$$

By measuring the angles $\theta_1$ and $\theta_2$, the range to the target can thus be calculated. These angles can be measured directly as radian measures from associated hardware or as the difference between a known reference, such as the centerline, of the two image planes (see Figure 2).

Relative depth between two different objects, rather than absolute depth, is easier to determine. From Figure 3, it can be seen that while it may be difficult to measure the actual angular disparity of points Q and P, it is easy to determine which object is closer by their relative displacement on the image planes $I_1$ and $I_2$ [Ref. 9:pp. 113-120 ; Ref. 10:pp. 158-161].

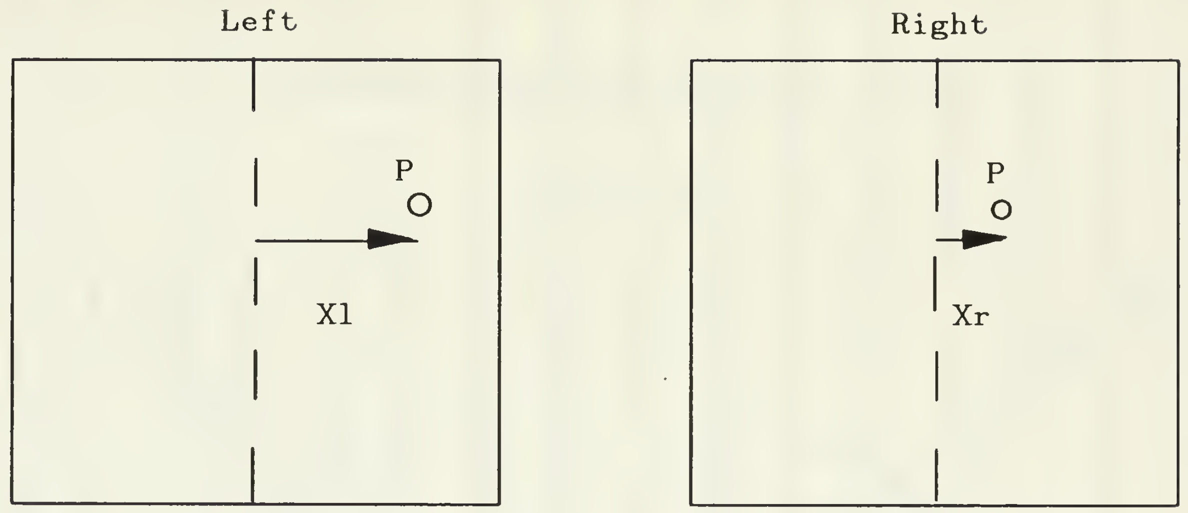

Figure 2.

Relative Left and Right Image Disparity

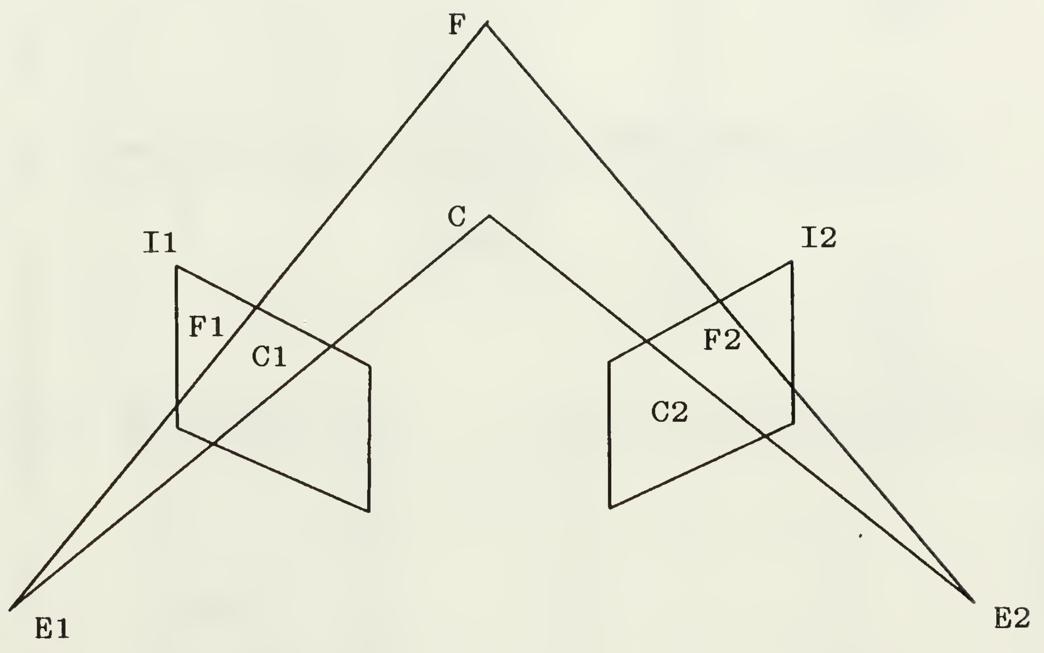

Figure 3.

Relative Depth From Binocular Vision

The theory of stereo disparity is simple and straightforward. However. implementing it is a difficult procedure and subject to significant error. There are two major techniques for implementing a depth measurement scheme via binocular vision. They are by passive and active means.

a. Passive Stereo Ranging

This approach is very similar to human vision. The distance to a point is inversely proportional to the disparity between the two image planes. The ranging becomes more accurate the farther apart the two cameras are or the closer the target is. If the two cameras are too far apart, though, the requirement for overlapping fields of view may become a difficult prerequisite to meet. Bringing the two cameras too close together makes the disparity between the two image planes very small and introduces error into the calculations.

The most significant problem with this technique is that a given point. P, has to be accurately identified in both images. If the location of P is displaced slightly due to thresholding or calibration discrepancies between the two cameras, the range determination will be in error. Moreover, just identifying P in both image planes is a problem. Since camera output is nothing more than a grey scale code of each pixel, and the position of P in each image plane will not be the same, all of the pixels in each image have to be correlated to each pixel in the other image. Segmentation of the image planes then has to be done and each segment has to be examined for areas of high correlation. Due to ambient lighting variations between the image planes, these correlation functions may be slightly in error and therefore the precise location of known points in each image that are also in the other image cannot be accurately determined [Ref. 11:pp. 15-23].

This technique is computationally intensive and results can only be considered accurate if a large number of correlated points in each image can be determined so that statistical methods of error reduction can be applied. However, its advantages lie in the fact that it is most similar to natural ranging methods and it is not power intensive. Illumination of the area of interest is supplied by nature, not by the autonomous vehicle. This natural illumination is also, though, the source of the most significant error since it cannot be preconditioned (an example would be imaging of a scene during night time with a lack of significant ambient light) and therefore is not considered well-behaved [Ref. 7:pp. 150-151 ; Ref. 8:pp. 111-114].

#### b. Active Stereo Ranging

A solution to the stereo image point correspondence problem is to provide an easy to identify point. With an optical sensor, that could be a point of light significantly brighter than the ambient light background. In this approach, a spot of light or a laser beam is scanned over a scene and the spot is tracked. The requisite angular information is measured in the normal way and range is computed similarly [Ref. 12:pp. 134-146]. The limiting factors in this ranging system are the inherent ability to measure small angles, the spot size of the beam, and the on board power requirements for the light source. Another method of active optical ranging was studied for a number of years at Rensselaer Polytechnic

Institute [Ref. 13]. This method did not depend on stereo sensors for it's triangulation to the points of interest but rather an array of directional photodiodes. These photodiodes would detect a laser beam reflection from the surrounding terrain which had been illuminated by a laser offset on a mast. By an appropriate set of transformations. the elevation and range to the point reflecting the laser could then be calculated.

A further enhancement is to project an entire line rather than a single spot. The line is scanned over the target and the reflected image is recorded by a camera. Points of interest can then be designated and triangulation methods can be applied to determine ranges in the scene. Surfaces that are nearly parallel to the illuminating plane. though, can return poor results. This can be overcome by projecting two mutually orthogonal series of scanning planes of light. Even further, a grid can be projected on the scene so that scanning hardware is eliminated [Ref. 14:pp. 150-151 : Ref. 15:pp. 112 ; Ref. 10:pp 167-169].

As the system is made more complicated, the processing requirements increase proportionately. Also. in any case, the inherent accuracy of the system is dependent on the ability to measure small angles precisely, just as before. As the range to the points of interest increases, the angles decrease and accurate range determination becomes more difficult. While the points of interest are more easily determined, this advantage is not without it's cost. The power required to illuminate the target is significant and, when operating in an environment where humans will be present. the distraction of illumination by an artificial grid may not be acceptable [Ref. 14:pp. 150-151 ; Ref. 12:pp. 112 ; Ref. 10:pp 167-169].

4. Optical Time of Flight

Optical time of flight combines many of the best characteristics of radar and of active stereo disparity and few of the disadvantages. Instead of using radio frequency electromagnetic energy, with it's associated long wavelength and pulse width, optical time of flight uses visible wavelength or near IR energy. Because of it's similarity to radar, optical time of flight range imaging is sometimes called LIDAR. This acronym stands for LIght Ranging and Detection.

Though, lidar operates in or near the visible band of electromagnetic radiation , it can be characterized by the same parameters as radar. It's beam width is calculated exactly as is a radar beam. Using Equation (2.2) and the typical operating wavelength of a GaAs diode laser of 850 nanometers, a 1 millimeter diameter focusing lens will produce a beam width of 1.04 milliradians [Ref. 16:pp. 156-167]. At 1000 meters (well beyond typical ranging distances for lidar) this beam has a beam width of 1.0 meters, much smaller than the corresponding radar beam. Attendant with the narrow beam width, the lidar has a significantly increased angular resolution when it is imaging a target and so it can generate a much more precise set of data.

Noise in the target area in the form of ambient light is a major source of error in binocular vision systems, both active and passive, as is radio noise a source of ranging error for radar. The narrow band-width of a laser beam when compared to the relatively wide band in which the power of natural visible light is distributed. reduces the noise problem enough so that. when measuring distance by time of flight, very narrow pulse widths can be used. For example. a one milliwatt helium-neon laser operating at 633 nanometers with a beam width of 1 milliradian and a band width of 0.2 nanometers has a spectral brightness of $1 \times 10^8$ lumens/$cm^2$-steradian-nm. The sun's spectral brightness is it's luminance ($1.5 \times 10^5$ lumens/$cm^2$-steradian) divided by it's band width (approximately 300 nanometers). This is about 500 lumens/$cm^2$-steradian-nm [Ref. 16:pp. 20-23]. The low power laser is over 5 orders of magnitude brighter than the sun, the major source of optical noise. Realization of this advantage is possible to some degree by using optical filters to eliminate all reflected light except that light in the lasers band width although available filter bandwidths are many orders of magnitude wider than laser bandwidths [Ref. 17:pp. 55-59].

Actually measuring range by a lidar can be accomplished by either pulsing the laser and measuring the time of reflection, as per radar, or by modulating a continuous wave laser beam and measuring the phase difference of the reflected beam. This second method has the added advantage of allowing doppler processing of the frequency of the reflected energy to derive velocity information on the target [Ref. 10:pp. 170 ; Ref. 14:pp. 151-152 ; Ref. 15:pp. 113-114].

Although the pulsed systems are more similar to radar, adequate range resolution and the minimum detectable ranges required by autonomous vehicles necessitate very short pulses. Pulses in the range of 10's of picoseconds will give range resolutions in the millimeter range [Ref. 14:pp. 151-152 : Ref. 15:pp. 113-114]. Picosecond pulses are possible using Q switching techniques, but that requires sophisticated and expensive equipment and the energy in such a small pulse is very low. Phase modulated CW ranging systems offer good range resolution and minimum range with less sophisticated and sensitive equipment. Modulation frequencies of 10 to 50 megahertz have been used to measure ranges with an accuracy of few millimeters over a range of less than a meter out to more than 10 meters [Ref. 10:pp. 170 ; Ref. 14:pp. 151-152 ; Ref. 15:pp. 113-114].

Optical time of flight range imaging systems are a relatively new technology. They promise good range resolution, very narrow beam-widths, good noise performance, and reasonable ranging distances for the local navigation problem. Unlike stereo optical techniques, the range resolution is not a function of the range from the sensor. Their beam-width is superior to conventional radar resulting in superior lateral resolution. As the technology matures, they may ultimately prove to be the dominant form of local range imaging for robotics.

## C. SYMBOLIC DESCRIPTION OF TERRAIN FOR AUTONOMOUS VEHICLES

Once the data from an imaging system has been collected, the next step for an autonomous vehicle is to glean information from it effectively. An array of grey scale pixels or x, y, z values is, in and of itself, rather useless. The generation of knowledge about the local environment from images is a two step process. These two steps have been appropriately titled as *early* and *late* visual processing [Ref. 9:pp. 87-160]. In early processing. the goal is to generate characteristic information about the scene that the late visual processing modules can use. Early processing has been the center of most vision research conducted until recently. The information generated by early processing modules is a line drawing termed the *intrinsic image* [Ref. 9:pp. 95]. The intrinsic image is defined as the grouping of the image into regions corresponding to physical surfaces. This segmentation is often done by identifying edge pixels in a scene (through edge identification algorithms which measure brightness gradients). connecting these edge pixels with lines (line generation algorithms) and then identifying the physical dimensions of the regions contained within the line boundaries. While this is the most common form of segmentation for the intrinsic image, there are as many different ways to segment a scene as there are people conducting research in machine vision. Each segmentation scheme amounts to a new concept in the effort to discover the inherent (or intrinsic) knowledge contained in the raw data obtained by optical sensors.

Once the intrinsic image has been defined. by whatever scheme. the next level in visual processing is called late image processing or sometimes image understanding [Ref. 9:pp. 87-160]. It is with the late image processing that artificial intelligence is most often associated. The difference between simply building and connecting lines and the act of discerning knowledge from those collections of lines is intelligence. The present level of intelligence programming on computers is at the level of symbolic computing. An intrinsic image is reduced to a symbolic description (length, width, color, height, radius of curvature, etc...) and then an attempt to match that symbolic description with a predefined set of known symbols is made. If a good match is made with symbol X, then the intrinsic image can be assumed to be another X or very similar to an X. Obviously, an autonomous vehicle cannot contain enough memory to store a complete set of symbols for every possible object in the world, and if it could, the processing time to search the entire set of symbols would be prohibitive. For an autonomous vehicle to navigate about a local environment, though, the required symbolic terrain data base can be made more reasonable.

## 1. Object Identification

There are two methods of image processing in the autonomous vehicle local navigation problem. These methods can be termed as *object identification* and *region classification* , respectively. The most common method is to attempt to identify individual objects in the local environment first. Then the path planner routines make appropriate decisions based on this knowledge [Ref. 18:pp. 223-235]. This object identification can be either on an extremely sophisticated level or on the simplest of levels.

A very simple system might only identify objects as obstacle or traversable space. This is the most elementary level at which vision works. The simplest way to think of the symbolic description of an obstacle is in terms of *voxel's.* A voxel (VOlume piXEL) is an incremental space in range and lateral displacement (dx, dy in cartesian coordinates) corresponding to the minimum range increment and horizontal separation detectable by the vision system. This is equivalent to a "range bin" in spherical coordinates. An obstacle will be then, a voxel with enough vertical development to impede the motion of the autonomous vehicle. This minimum development may be anything greater than 1 inch for a small wheeled vehicle, for instance. (A more sophisticated system might have to concern itself with vertical development down from the ceiling for a tall vehicle. An example of this would be a doorway.) Voxels that are not obstacles are then classified as traversable. Their symbolic description would be characterized as a lack of assigned attributes [Ref. 18:pp. 223-235].

Obstacle elements that are adjacent can be joined to each other to create the intrinsic image [Ref. 19:pp. 119-123]. Any traversable spaces that are completely enclosed by obstacles should then be classified as obstacles also. For this simple vision system, after the scene has been segmented into obstacle and traversable space, the autonomous vehicle need go no further in object recognition. The symbolic description of the scene would be groups of obstacles, each one specified by a position and shape. In this case, the object identification routines are not concerned with questions about the physical nature of the obstacles. It makes no difference if the obstacle is a desk or a wall. Indeed, the obstacle may be a desk against a wall, for all the system cares, and it will be treated the same.

For most autonomous systems. the above level of sophistication is not sufficient. It only provides a vehicle with the ability to avoid collisions with obstacles. It cannot provide the vehicle with sufficient knowledge of a scene for the vehicle to identify specific objects in that scene. The more the autonomous vehicle is required to exhibit intelligent behavior in it's interaction with the local environment, the more the symbolic description of the scene will have to be detailed. Absolute precision in the symbolic description of the terrain will seldom, if ever. however. be obtainable. This is due to the problem of self occlusion. The reverse side of an object is hidden from imaging systems and objects in front of others will hide detail. This has lead to range images being classified as 2 1/2 dimensional versus 3 dimensional [Ref. 10:pp 158 ; Ref. 20:pp. 69]. Thus, any symbolic description of an object will be partial at best and will cause errors when matching it with known object symbolic descriptions.

Once an intrinsic image has been created, inherent characteristics of an object may be derived [Ref. 21:pp. 3-4]. Those inherent characteristics comprise what will be called the *symbolic description* of that object [Ref. 9:pp. 143-160]. These characteristics will consist of such things as the x, y, and z dimensions of the object. for instance. Derivation of these characteristics will allow an object's parameters. such as its volume and its major and minor axis, to be calculated. Radius of curvature is also a characteristic which is easily derivable from the intrinsic image. The choice of the characteristic a system attempts to derive from the intrinsic image is driven by the type of symbols that are chosen to describe the objects. Some algorithms attempt to describe objects in terms general parallelograms (x,' y. and z coordinates) while others use generalized cylinders (radius of curvature) [Ref. 22:pp 629-637 : Ref. 19:pp. 119-123]. These two characteristics have been chosen only for the purposes of illustration. Algorithms have been utilized which measure the smoothness of a surface. moments of the shapes. and many many more [Ref. 5:pp. 1-15].

The derivation of all of the above types of information would be extremely expensive in terms of computational time. For this reason, most vision research attempts to identify a few important inherent characteristics to analyze and then to optimize the algorithms used to produce those characteristics or implement the routines in hardware. If the number of individual objects a system has to identify is small, then the number of symbolic descriptors required can be kept proportionately small and the computational time required to generate them can be kept manageable. However, for real world applications, most environments that an autonomous system will have to navigate in will be complicated scenes. Consider an office space, a factory or a typical home, for instance. The number of individual objects present in the environment at any one time will be large and, even for an algorithm that generates the minimum number of symbolic descriptors, the computational time required to fully describe all the objects in the scene will be very large.

### 2. Region Classification

For an autonomous vehicle to navigate in a local environment, it may not be necessary for the system to actually identify each element in the scene. For example, in an exterior environment, it is not necessary for a robot to differentiate between a telephone pole and a tree. It is enough to recognize the presence of an obstacle. However, the simple obstacle recognition systems are not sufficient either. An autonomous vehicle has the ability to navigate over a variety of terrain types and the vehicle may have to make a decision on the basis of a cost function of which type of terrain to travel over. An example would be a scene with a small hill directly in the vehicle's path and flat terrain off to the side. The vehicle would have to decide on traveling over the hill or altering it's path enough to follow the flat terrain. For this type of vision another approach to object recognition, termed *region classification*, has recently been proposed [Ref. 23:p. 1146].

The concept of region classification is introduced to avoid the time consuming and complex process of object identification by using topographic features as the basis for an autonomous vehicle's navigation decisions. Topographic features can be discerned rather rapidly based on the gradient and second directional derivatives around an individual voxel. These topographic features include holes and poles, valleys and ridges, flat and inclined (safe and unsafe inclined) terrain, and saddles [Ref. 24:pp. 22-24]. For an autonomous vehicle, this vision technique would classify both a tree and a telephone pole as, simply, a pole. This eliminates the extra time required to distinguish, on the basis of more complex symbolic descriptors between the two objects. Once a voxel has been classified, a symbolic map of the local terrain can be constructed for use by the path planning module. This process will be more completely described later in this thesis.

## D. SPECIAL REQUIREMENTS OF WALKING VEHICLES

An autonomous vehicle may be described by a number of different classification parameters, all of which have a significant effect on the vehicle's performance. The mobility of a vehicle is typically classified into two major areas. These are wheeled and legged systems [Ref. 25:pp. 1-15,pp. 73-74]. The wheeled systems may be further divided into conventional wheeled and tracked systems. Each type of system has it's strong points and it's deficiencies.

Wheeled and tracked systems have a preferred direction of motion. In a well defined environment, such as an interior space, directional preference may prove to be only a small deficiency or even an advantage. Having a preferred direction limits the vision requirements of the system because the system can safely concentrate it's vision in the forward direction. With the ability to retain a model of the space it passes over, when reverse motion is required, due to an error in navigation, a reverse vision capability is not an absolute requirement. This reduction in the vision system requirements limits the complexity of the system. Also, by virtue of the fact that wheeled systems have been utilized for a very long time, such systems are simpler to design and build.

However. in a poorly defined environment. such as natural terrain. a wheeled vehicle is at a definite disadvantage when compared to a legged vehicle [Ref. 12:p. 134]. Legged vehicles have no preferred direction of motion. They may translate sideways with as much ease as maneuvering forward or backward. This allows a significant increase in maneuverability of the vehicle. Additionally, whereas a wheeled vehicle is limited in its step crossing ability by the diameter of its wheels and the bottoming of its body, a legged vehicle is only limited by the working envelope of its legs. This working envelope is defined by the physical length of the legs and the requirement to maintain the center of gravity within the support base of the legs in physical contact with the ground [Ref. 25:pp. 73-74]. Clearly. a properly designed legged vehicle will have a much greater step crossing ability than a similar wheeled vehicle.

This increase in maneuverability is not without penalties, though. First, because the mechanics of designing and building legged systems is not well understood, systems development takes extensive time and requires significant resources. It is only within the last twenty or so years that experiments with such systems have been possible. For the most part. these vehicles have been slow and have had a rough ride. Within the last decade, the first few legged systems starting to exhibit an ability to simulate biological motion have been built and demonstrated [Ref .18:pp. 7-15]. Research in this area has been slow, but recently has been showing significant progress.

Secondly, the inherent increase in mobility associated with legged vehicles leads to another penalty by putting significant demands on the associated vision systems. Since such systems can move sideways as well as forwards and backwards, a vision system designed to work with an independent autonomous legged system should be able to preview the terrain to both sides as well as fore and aft. Additionally, depending on the type of terrain a vehicle is crossing, the vision system must be able to classify surface variations of much smaller magnitude than presently required. Specifically, with the level of sensing demonstrated to date for walking vehicles, computer-controlled limb cycling in response to human commanded vehicle direction and speed is sufficient to maneuver a walking vehicle over smooth terrain only. However, because a hole as small as the foot pad could prove to be a potentially destructive obstacle, a vision system must be able to detect such an object and to accurately sense its position [Ref. 12:pp. 134-135]. Also, in order to traverse the rough terrain these legged vehicles are envisioned for, they must be able to "see" individual foot holds for the navigation and control system to utilize; otherwise their mobility will only be marginally better than wheeled systems.

## E. APPROACHES TO FEATURE EXTRACTION

### 1. Robot Rover

One of the earliest attempts to combine vision with an autonomous vehicle was the *Robot Rover* project by Hans P. Moravec at the Stanford AI lab

[Ref. 11]. This vehicle used stereo vision to generate a three dimensional view of the local environment. One of the unique aspects of this stereo vision system was that it was implemented with a single camera. The camera was mounted on a 52 centimeter slider bar above a simple mechanized cart. At each sampling point the cart would stop and the camera would slide from left to right taking nine pictures at 6.5 centimeter intervals. An interest operator would then identify approximately thirty points in the fifth (middle) image for a correlator module to attempt to match in the other images. Once the points were matched, their range could be calculated, off cart, on a DEC KL10 computer conected by means of a radio link, using the apparent parallax between images. The reason for using nine images was to attempt to reduce the effect of noise in generating range errors since the correlator would often identify features incorrectly. Specifically, in Moravec's work, the 36 combinations in which the nine images could be combined were plotted to produce a histogram of matches and ranges, in which the effect of a few incorrect matches were typically outweighed by the true peaks.

The navigation algorithm for the cart depended solely on the vision system for it's inputs. After the cart took it's first view of the local environment, it would move forward about one meter and take another look. The difference between the two views of the position of fixed objects was the actual movement of the cart. The vehicle would then plan the next move, move one meter, and then look again. Each movement cycle took approximately fifteen minutes.

Problems with the system limited it's effectiveness. Firstly. the vision system could not reliably identify features in a relatively bland environment of primarily simple polygons. Obstacles which lacked sufficiently high contrast were difficult for the correlation operator to accurately localize. This had the effect of introducing significant noise into the system. Secondly, the self navigation module was very sensitive to the noise effect. The cart motion was solved for by finding a 3D translation and rotation which best fit the before and after scenes. The small errors in position between the two sets of images would force the system to converge on an improper transform. introducing a navigational error into the system. After a positional error was introduced into the system, the path planner had inaccurate data on which to act and ultimately, the system would get lost and run into an obstacle.

### 2. OSU Hexapod

Whereas the Robot Rover made use of a passive triangulation system, the OSU Hexapod used an active triangulation system for vision [Ref. 12:pp.134-137, pp. 143-145]. The purpose of this vision system was not for navigation or local terrain identification but, instead, to designate footholds for the hexapod to utilize.

A legged vehicle, as was discussed before, must be able to identify individual footholds as it moves along a path. Over smooth terrain this is not critical. However, over rough terrain, without terrain preview information, the vehicle is unable to safely move (except at very low speeds) since a mispositioning of one of the footpads could be catastrophic. Lacking an intelligent vision system for the OSU Hexapod, a human operator had to designate footholds to the hexapod. This was done by means of a laser. The operator would pick out likely footholds within the range of motion of the legs and illuminate them with the laser. Two charge-injection-device (CID) cameras were mounted on masts above the vehicle with fields of view looking forward and down. Special hardware was built to determine to position of the designated point in both camera's fields of view and to transmit that information to a PDP-11/70 computer to calculate the range and bearing.

In an indoor environment, this vision system proved to be effective within the limits of the system. Errors in the position determination were 1.34 cm forward, 1.16 cm lateral, and 1.2 cm vertical. The half-cycle time of motion was 25 seconds, allowing the operator sufficient time to designate the next foothold and the system time to accept or reject it. The motion was normally smooth with no noticeable delay. However, when the system was exposed to an environment with a large amount of background noise in the form of ambient light, the system had difficulty distinguishing the designated spot. The laser used to designate the footholds was a five milliwatt helium neon laser. Apparently, this was not enough power to provide adequate contrast with the ambient light.

### 3. Autonomous Land Vehicle

The present state of the art in range imaging is represented by the Autonomous Land Vehicle (ALV) [Ref. 26:pp. 1-26]. The ALV, one of the main application areas for DARPA's Strategic Computing program. is a program that combines recent advances in sensors. vision. reasoning and advanced computer architectures with a mobile 8-wheeled vehicle in an attempt to advance and demonstrate autonomous navigation and decisionmaking. The test site for the ALV encompasses nearly every kind of terrain an autonomous vehicle could be envisioned encountering. Initial testing will only be on roads but will eventually progress to include open terrain. At present the system is required to negotiate a two kilometer asphalt road at speeds up to ten kilometers per hour [Ref. 26:pp. 8,16].

The vision system for the ALV is a combination of an RCA color video CCD matrix TV camera and the ERIM laser range scanner [Ref. 26:pp. 19-23]. The vision system switches between the two imaging systems based on the requirements of the reasoning subsystem. The processing algorithm for the video subsystem detects the difference in color between the road surface and the shoulder and vegetated areas. The ranging system works analogously to the video subsystem except, instead of looking for a color contrast, the ranging systems looks for a contrast in the smoothness of the terrain. This contrast in the smoothness of the terrain results from the road being paved and therefore much smoother than the shoulder which includes small rocks, potholes and ditches [Ref. 5:pp.4-15]. The advantage of the ranging system over the video system is that the ranging system will work over significantly greater variations in lighting and weather conditions (day, night, and even when the ground is covered by a layer of snow). While an algorithm measuring smoothness can do things like find obstacles alongside the road. the algorithm. being inherently non-responsive to intensity or color. could not attempt to follow a painted line on the road (unless reflectance data is also used). Also, while the video subsystem can generate an image in about 1/30th of a second, the ranging subsystem can only generate an image about every 1/2 second. Thus the two subsystems, video and ranging, complement each other to make a more effective real time vision system.

### 4. Region Classification

In 1985, Olivier [Ref. 24:pp. 12-44], proposed using the concept of terrain or region classification based on the output of a laser range finder. His method was not implemented in hardware but was, rather, simulated on a PDP-11/70 computer. In his approach, he classified each terrain cell as one of 7 types: peak, pit, ridge, ravine, flat, safe slope, and unsafe slope regions.

In his investigation, Olivier first simulated the input from a range scanner in terms of a matrix of elevation data versus latitude and longitude. Then a quadratic equation was fit to the local region about each surface voxel. The gradient vector and Hessian matrix at each such voxel was then calculated. Finally, this data was used to generate a terrain classification for each voxel in the elevation map. From the results of this terrain classification, his algorithm would print out a symbolic map of the local terrain.

It should be pointed out here that Olivier's masters thesis was specifically concerned with a navigation algorithm to do path planning for an autonomous vehicle. As such his development of the region classification algorithm was limited in scope and depth.

## F. SUMMARY

Vision research for autonomous vehicles has been a primary focus of robotic research for a number of years. The vision research has progressed from simple systems requiring long periods of time to generate a single images to almost real time. very accurate, complex systems generating large amounts of data on a large scale scene. For the navigation problem, range data offers one particular data format that can be used to simplify the processing requirements of the on board computers. Instead of trying to derive individual object analysis and identification from the intrinsic image, the range data format can be utilized to do region classification on local terrain with each voxel being one classified region. In regards to the range imaging systems, the scanning laser range finder seems to offer better performance than most other systems developed to date and is presently the object of significant research.

However, any of the methods for image analysis discussed in this chapter are as dependent on mathematical concepts as on hardware. In the next chapter, a detailed analysis of the mathematical concepts behind the region classification method of image analysis, as origionally proposed by Olivier and then further extended by the author, will be presented since this scheme is particularly applicable to the laser scanner employed on the OSU ASV.

# III. MODELS FOR TERRAIN GENERATION AND CLASSIFICATION

## A. INTRODUCTION

The first step in developing any algorithm to handle a problem such as region classification, is to define the limits of the problem. In this case, the purpose of the algorithm is to classify the visible terrain surrounding an autonomous walking vehicle. Therefore, a terrain data base has to be available to the algorithm. This could be a real terrain scene that is imaged by an actual laser scanner with the output being recorded for future use or the data base could be an artificial terrain scene generated and made available to the computer program. For the purposes of this thesis, it was decided to simulate the terrain using the available computer resources.

Once the terrain has been generated, an appropriate set of rules has to be developed to allow the program to classify images in the field of view. These rules will be derived, in part, from the earlier work of Olivier [Ref. 23:pp. 1-6 ; Ref. 24:pp. 21-44] who developed a mathematical method for obtaining the characteristic first and second derivatives of individual voxels in a range image. In this chapter, the mathematical concepts behind his algorithm will be developed in more detail and then expanded on.

## B. GENERATION OF SYNTHETIC TERRAIN

There are any of a number of methods to mathematically generate equally acceptable synthetic terrain for this simulation. The choice of methods depends on whether a *deterministic* or a *random* form of terrain is desired. Deterministic means that the prominent terrain feature of the scene is predetermined and then the terrain is mathematically generated to produce a model of that feature. On the other hand, generation of random terrain implies the use of random number generators which produce entirely random prominent terrain features. A deterministic model is useful in testing the simulation against specific terrain types, such as an isolated hill, valley, or cliff terrain feature which is then not effected by the presence of other nearby terrain features. A random terrain model, however, is useful in determining the overall simulation performance in a real world environment in which few, if any, terrain features are independent of the other surrounding terrain features.

### 1. Generation of Deterministic Terrain

There are a number of possible methods used to generate deterministic terrain. The easiest method is to simply have the person running the simulation input the x, y, and z values of the terrain to be simulated into a matrix which can then be referenced by the simulation. This approach is appropriate for small areas of interest, but for the simulation of a large physical area it rapidly becomes unmanageable.

To automatically generate a terrain simulation for a large physical area, it is necessary to use some form of mathematical formula and then have the computer calculate the terrain matrix. The formula could be, for example, a Fourier series, an appropriately sized polynomial function, or a cylindrical function. To use a Fourier series to simulate a specific three dimensional feature, the user must derive the coefficients for each of the sinusoids used in the Fourier series [Ref. 27:pp. 11-46]. For a simple two-dimensional feature, deriving the Fourier coefficients is relatively straightforward. However, for a more complicated three-dimensional feature the derivation of the coefficients could prove to be very difficult. To use a polynomial series would similarly require the user to determine the coefficients of for each member of the series, an exhaustive and time consuming process.

The idea of a cylindrical function is a variation of the three dimensional polynomial in which a two dimensional function is derived to simulate the desired terrain feature and is then expanded linearly along the third dimension or axis [Ref. 28 :pp. 34-36]. This produces a stair step effect and is adequate to reproduce terrain features such as "rolling" terrain and objects such as linear "ridge lines" or "valleys". Unfortunately, features such as hills or depressions can not be modeled by use of a cylindrical function.

### 2. Generation of Random Terrain

One cannot simply use the random number generator to generate random elevation coordinates for random terrain because the results are simply a noisy but flat terrain with no prominent terrain features. Instead, one of the previously mentioned techniques is used with the exception that the coefficients, which determine the behavior of the function, are chosen at random. If the form of a polynomial function, of sufficiently low order to keep the computational time required to a minimum, is chosen, then the result will be a terrain scene which over a large physical area will not be very realistic. For example, a function of order two will always be a parabola which can be a reasonable local simulation of a hill or a depression but fails over a large area because the resulting parabola will be infinitely tall or deep, unlike real terrain features.

A reasonable simulation is achieved by use of a function similar to a Fourier series. In a true Fourier series, the sinusoids are based on the fundamental frequency and it's harmonics. This produces a terrain which is regular and repetitive. For example, if only the fundamental frequency is used, the terrain simulation appears to be a series of waves emanating from one central point, just like the waves a water drop makes when dropped on a calm surface. Again, the results are usable but not very realistic. A variation of the Fourier series which produces more realistic results is the use of a series of sinusoids which, however, have random origins, amplitudes, and frequencies. This produces highly variable terrain which can simulate many normal terrain features. The added advantage of using trigonometric functions in generating the terrain scenes is that, unlike orthonormal polynomials which can also accurately simulate physical terrain, the trigonometric functions have properties which are well known and are installed in libraries on most computer systems.

### 3. Methods Used in Simulation

For the purposes of this simulation, it was decided to use two of the previously discussed methods to generate the test terrain for the simulation. As the algorithm was developed, it was tested against a deterministic model. This was a local neighborhood test only (8 nearest neighbors) and so a large number of terrain models could be manually input as data. The algorithm could then be validated against a number of well defined terrain types. When the full simulation is run, though, the area to be analyzed is a 30 distance unit by 30 distance unit area (30x30 matrix) which is clearly too large to be input manually. Therefore the simulation uses the variation of the Fourier series in which the parameters are randomized. This produces a large variation in the terrain within each scene and is therefore a more realistic model.

## C. QUADRATIC APPROXIMATION OF SURFACE BY LEAST SQUARES FIT

In his thesis. Olivier bases his work on first fitting a least squares quadratic surface to each surface voxel and it's eight nearest neighbors. Thus. around each such voxel, the elevation (z) of each point in the area of interest is a function of the x and y coordinates:

$$z = f(x,y) \tag{3.1}$$

(For this derivation a standard right handed coordinate system is used. The x coordinate is positive to the right, y is positive in a $-90^{o}$ direction measured in the counter-clockwise direction and z (elevation) is positive up.) A quadratic fit as a function of x and y has the following form:

$$\hat{f}(x,y) = k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 \tag{3.2}$$

The square of the error between the fitted equation and the real terrain is then defined as

$$\epsilon^2 = \sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 - f(x,y))^2 \tag{3.3}$$

The partial of the squared error with respect to each constant, k .can then be taken as:

$$\frac{\partial\epsilon^2}{\partial k_1} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 - f(x,y))\right] 1 \tag{3.4}$$

$$\frac{\partial\epsilon^2}{\partial k_2} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 - f(x,y))\right] x \tag{3.5}$$

$$\frac{\partial\epsilon^2}{\partial k_3} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 - f(x,y))\right] y \tag{3.6}$$

$$\frac{\partial\epsilon^2}{\partial k_4} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 - f(x,y))\right] x^2 \tag{3.7}$$

$$\frac{\partial \epsilon^2}{\partial k_5} = \left[ 2\sum_x\sum_y (k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 - f(x,y)) \right] xy \qquad (3.8)$$

$$\frac{\partial \epsilon^2}{\partial k_6} = \left[ 2\sum_x\sum_y (k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 - f(x,y)) \right] y^2 \qquad (3.9)$$

To minimize the squared error, each of the above equations is expanded, the factor of two is divided out and then the partials are set to zero. Thus,

$$0 = \sum_x\sum_y k_1 + \sum_x\sum_y k_2x + \sum_x\sum_y k_3y + \sum_x\sum_y k_4x^2$$
$$+ \sum_x\sum_y k_5xy + \sum_x\sum_y k_6y^2 - \sum_x\sum_y f(x,y) \qquad (3.10)$$

$$0 = \sum_x\sum_y k_1x + \sum_x\sum_y k_2x^2 + \sum_x\sum_y k_3xy + \sum_x\sum_y k_4x^3$$
$$+ \sum_x\sum_y k_5x^2y + \sum_x\sum_y k_6xy^2 - \sum_x\sum_y f(x,y)x \qquad (3.11)$$

$$0 = \sum_x\sum_y k_1y + \sum_x\sum_y k_2xy + \sum_x\sum_y k_3y^2 + \sum_x\sum_y k_4x^2y$$
$$+ \sum_x\sum_y k_5xy^2 + \sum_x\sum_y k_6y^3 - \sum_x\sum_y f(x,y)y \qquad (3.12)$$

$$0 = \sum_x\sum_y k_1x^2 + \sum_x\sum_y k_2x^3 + \sum_x\sum_y k_3x^2y + \sum_x\sum_y k_4x^4$$
$$+ \sum_x\sum_y k_5x^3y + \sum_x\sum_y k_6x^2y^2 - \sum_x\sum_y f(x,y)x^2 \qquad (3.13)$$

$$0 = \sum_x\sum_y k_1xy + \sum_x\sum_y k_2x^2y + \sum_x\sum_y k_3xy^2 + \sum_x\sum_y k_4x^3y$$
$$+ \sum_x\sum_y k_5x^2y^2 + \sum_x\sum_y k_6xy^3 - \sum_x\sum_y f(x,y)xy \qquad (3.14)$$

$$0 = \sum_x\sum_y k_1 y^2 - \sum_x\sum_y k_2 x y^2 - \sum_x\sum_y k_3 y^3 - \sum_x\sum_y k_4 x^2 y^2$$
$$- \sum_x\sum_y k_5 x y^3 - \sum_x\sum_y k_6 y^4 - \sum_x\sum_y f(x,y) y^2 \quad (3.15)$$

In order to simplify these equations so that they may be more easily adapted for used by the algorithm an assumption has to be made. This is based on the form in which the coordinate data is formatted. The coordinates of each voxel in the 3x3 window being modeled are measured relative to the center voxel (the voxel of interest). Using this format the coordinates of each of the voxel's is simply ± one distance unit from the voxel of interest. This has the added benefit of non-dimensionalizing the entire algorithm so that it can be applied to scenes of any physical size. TABLE 1 shows the resultant relative x and y coordinates of the 3x3 window. Using TABLE 1, exhaustive calculation shows that the following identities are true for $n \leqslant 3$ :

$$0 = \sum_x\sum_y x^{(2n+1)} \quad (3.16)$$

$$0 = \sum_x\sum_y y^{(2n+1)} \quad (3.17)$$

$$0 = \sum_x\sum_y x^{(2n)} y^{(2n+1)} \quad (3.18)$$

$$0 = \sum_x\sum_y x^{(2n+1)} y^{(2n)} \quad (3.19)$$

$$0 = \sum_x\sum_y x^{(2n+1)} y^{(2n+1)} \quad (3.20)$$

In other words, any terms in Equations 3.10 - 3.15 which contain a summation of an x or y coordinate with an odd power will identically go to zero.

TABLE 1

RELATIVE DISTANCE
BETWEEN VOXEL'S

| | | |
|---|---|---|
| x=-1 y= 1 | x= 0 y= 1 | x= 1 y= 1 |
| x=-1 y= 0 | x= 0 y= 0 | x= 1 y= 0 |
| x=-1 y=-1 | x= 0 y=-1 | x= 1 y=-1 |

Accordingly, Equations 3.10 - 3.15 can be rewritten as:

$$0 = k_1\sum_x\sum_y 1 + k_4\sum_x\sum_y x^2 + k_6\sum_x\sum_y y^2 - \sum_x\sum_y f(x,y) \tag{3.21}$$

$$0 = k_2\sum_x\sum_y x^2 - \sum_x\sum_y f(x,y)x \tag{3.22}$$

$$0 = k_3\sum_x\sum_y y^2 - \sum_x\sum_y f(x,y)y \tag{3.23}$$

$$0 = k_1\sum_x\sum_y x^2 + k_4\sum_x\sum_y x^4 + k_6\sum_x\sum_y x^2y^2 - \sum_x\sum_y f(x,y)x^2 \tag{3.24}$$

$$0 = k_5\sum_x\sum_y x^2y^2 - \sum_x\sum_y f(x,y)xy \tag{3.25}$$

$$0 = k_1 \sum_x \sum_y y^2 - k_4 \sum_x \sum_y x^2 y^2 - k_6 \sum_x \sum_y y^4 - \sum_x \sum_y f(x,y) y^2 \qquad (3.26)$$

By again referring to TABLE 1, it is found that:

$$\sum_x \sum_y x^2 = 6 = \sum_x \sum_y y^2 \qquad (3.27)$$

$$\sum_x \sum_y x^4 = 6 = \sum_x \sum_y y^4 \qquad (3.28)$$

$$\sum_x \sum_y x^2 y^2 = 4 \qquad (3.29)$$

$$\sum_x \sum_y 1 = 9 \qquad (3.30)$$

Combining Equations 3.22 and 3.27, $k_2$ is found to be:

$$k_2 = \frac{\left[\sum_x \sum_y x f(x,y)\right]}{\left[\sum_x \sum_y x^2\right]} = \frac{\left[\sum_x \sum_y x f(x,y)\right]}{6} \qquad (3.31)$$

Combining Equations 3.23 and 3.27, $k_3$ is found to be:

$$k_3 = \frac{\left[\sum_x \sum_y y f(x,y)\right]}{\left[\sum_x \sum_y y^2\right]} = \frac{\left[\sum_x \sum_y y f(x,y)\right]}{6} \qquad (3.32)$$

And finally, combining Equations 3.25 and 3.29, $k_5$ is found to be:

$$k_5 = \frac{\left[\sum_x \sum_y xyf(x.y)\right]}{\left[\sum_x \sum_y x^2 y^2\right]} = \frac{\left[\sum_x \sum_y xyf(x.y)\right]}{4} \tag{3.33}$$

Furthermore, combining Equations 3.21, 3.24, and 3.26 with Equations 3.27 - 3.30 and then rearranging the resulting equations slightly by adding the terms containing $f(x,y)$ to both sides of the equations produces the following three equations, respectively:

$$9k_1 + 6k_4 + 6k_6 = \sum_x \sum_y f(x,y) \tag{3.34}$$

$$6k_1 + 6k_4 + 4k_6 = \sum_x \sum_y f(x,y)x^2 \tag{3.35}$$

$$6k_1 + 4k_4 + 6k_6 = \sum_x \sum_y f(x,y)y^2 \tag{3.36}$$

Dividing Equation 3.34 by 3 and Equations 3.35 and 3.36 by 2 produces the following equations:

$$3k_1 + 2k_4 + 2k_6 = \frac{1}{3} \sum_x \sum_y f(x,y) \tag{3.37}$$

$$3k_1 + 3k_4 + 2k_6 = \frac{1}{2} \sum_x \sum_y f(x,y)x^2 \tag{3.38}$$

$$3k_1 + 2k_4 + 3k_6 = \frac{1}{2} \sum_x \sum_y f(x,y)y^2 \tag{3.39}$$

For the purposes of this algorithm. finding the value of the constant $k_1$ is not important. (However. to validate the formula's derived in this section. it is essential to calculate the value of $k_1$. This leads to an alternate derivation. using matrix algebra methods. of a matrix equation to calculate all the coefficients $k_1$ to $k_6$. The method presented in this section reproduces the results obtained by Olivier. The alternate derivation is presented in Appendix A. The section on the validation of the formula's is presented later in this chapter.) Accordingly. $k_1$ can be eliminated from the above equations by subtracting Equation 3.37 from Equations 3.38 and 3.39 respectively. Subtracting Equation 3.37 from Equation 3.38 yields:

$$k_4 = \frac{1}{2} \sum_x \sum_y f(x,y) x^2 - \frac{1}{3} \sum_x \sum_y f(x,y) \tag{3.40}$$

Subtracting Equation 3.37 from Equation 3.39 yields:

$$k_6 = \frac{1}{2} \sum_x \sum_y f(x,y) y^2 - \frac{1}{3} \sum_x \sum_y f(x,y) \tag{3.41}$$

So far, this analysis has yielded 5 easily solvable independent equations (reiterated below as Equations 3.31-33 and 3.40-41) which will generate 5 of the 6 constants necessary to fit a quadratic equation to a 3x3 window of a range image based on a least squares error analysis; the $6^{th}$ constant, $k_1$, is deemed inconsequential to the algorithm.

$$k_2 = \frac{\left[\sum_x \sum_y x f(x,y)\right]}{6} \tag{3.31}$$

$$k_3 = \frac{\left[\sum_x \sum_y y f(x,y)\right]}{6} \tag{3.32}$$

$$k_4 = \frac{1}{2} \sum_x \sum_y x^2 f(x,y) - \frac{1}{3} \sum_x \sum_y f(x,y) \tag{3.40}$$

$$k_5 = \frac{\left[\sum_x \sum_y xy f(x,y)\right]}{4} \tag{3.33}$$

$$k_6 = \frac{1}{2} \sum_x \sum_y y^2 f(x,y) - \frac{1}{3} \sum_x \sum_y f(x,y) \tag{3.41}$$

The purpose of this algorithm is not to model the terrain window but rather to analyze the terrain window based on easily definable characteristics of the window. These characteristics will be the slope and the curvature of the function fitted to the window. These characteristics will be generated using the Gradient and the Hessian matrix of the function fitted to the terrain window. Referring to Equation 3.2 the following quantities can be found. The first partial derivative of $f(x,y)$ with respect to x is:

$$\frac{\partial f(x,y)}{\partial x} = k_2 + 2k_4 x + k_5 y \tag{3.42}$$

and the first partial derivative of $f(x.y)$ with respect to y is:

$$\frac{\partial f(x.y)}{\partial y} = k_3 - k_5 x - 2k_6 y \tag{3.43}$$

The second partial of $f(x,y)$ with respect to x and y is:

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} = k_5 = \frac{\partial^2 f(x.y)}{\partial y \partial y} \tag{3.44}$$

The second partial of $f(x,y)$ with respect to y is:

$$\frac{\partial^2 f(x,y)}{\partial y^2} = 2k_6 \tag{3.45}$$

and the second partial of $f(x,y)$ with respect to x is:

$$\frac{\partial^2 f(x,y)}{\partial x^2} = 2k_4 \tag{3.46}$$

The gradient of a function is defined to be [Ref. 29:pp. 597]:

$$\nabla f(x,y) = \frac{\partial f(x.y)}{\partial x} \vec{\imath} + \frac{\partial f(x,y)}{\partial y} \vec{\jmath} \tag{3.47}$$

which from Equations 3.42 and 3.43 is:

$$= \left( k_2 + 2k_4 x + k_5 y \right) \vec{\imath} + \left( k_3 + k_5 x + 2k_6 y \right) \vec{\jmath} \tag{3.48}$$

Referring again to TABLE 1. the gradient of the voxel of interest is at the point x=0 and y=0. This simplifies Equation 3.48 to be:

$$\nabla f(x,y) = k_2 \vec{\imath} + k_3 \vec{\jmath} \tag{3.49}$$

Moreover. it can be proved [Ref. 29:p. 597] that the absolute value of the magnitude of the Gradient of the voxel is equal to the maximum slope at that point. Therefore the maximum slope of the voxel is defined to be simply:

$$slope_{\max} = \left( k_2^2 + k_3^2 \right)^{\frac{1}{2}} \tag{3.50}$$

The Hessian matrix is defined to be [Ref. 30:pp. 249]:

$$H = \begin{bmatrix} \dfrac{\partial^2 f(x,y)}{\partial y^2} & \dfrac{\partial^2 f(x,y)}{\partial x \partial y} \\ \dfrac{\partial^2 f(x,y)}{\partial y \partial x} & \dfrac{\partial^2 f(x,y)}{\partial x^2} \end{bmatrix} \tag{3.51}$$

By substituting Equations 3.44 - 3.46 into Equation 3.51, the Hessian is found to be simply:

$$H = \begin{bmatrix} 2k_6 & k_5 \\ k_5 & 2k_4 \end{bmatrix} \tag{3.52}$$

For reasons that will be detailed later in this thesis, it is necessary to determine the eigenvalues of the Hessian Matrix to properly classify a terrain voxel. The eigenvalues, $\lambda_1$and$\lambda_2$, of the matrix are defined such that [Ref. 31:p. 3]:

$$\left| H - \lambda I \right| = 0 \tag{3.53}$$

Using Equation 3.52, this means that:

$$0 = \begin{vmatrix} 2k_6 - \lambda & k_5 \\ k_5 & 2k_4 - \lambda \end{vmatrix} \tag{3.54}$$

or, solving for the determinant:

$$(2k_6 - \lambda)\,(2k_4 - \lambda) - k_5^2 = 0 \tag{3.54}$$

Expanding Equation 3.54 algebraically and grouping terms produces:

$$\lambda^2 + (-2k_4 - 2k_6)\lambda + (2k_4 2k_6 - k_5^2) = 0 \tag{3.55}$$

Using the quadratic equation, the roots of Equation 3.55, the eigenvalues, can be found as:

$$\lambda = \frac{(2k_4 + 2k_6) \pm \sqrt{(-2k_4 - 2k_6)^2 - 4(2k_4 2k_6 - k_5^2)}}{2} \tag{3.56}$$

For the purposes of this discussion, let the larger magnitude eigenvalue be $\lambda_1$ and the smaller one be $\lambda_2$. To be able to apply these results to a real terrain elevation image, it is necessary to show that the quantity under the radical is always positive and therefore the eigenvalues are non-imaginary. This is true if:

$$(-2k_4 - 2k_6)^2 \geqslant 4(2k_4 2k_6 - k_5^2) \tag{3.57}$$

or

$$4k_4^2 + 8k_4k_6 - 4k_6^2 \geqslant 16k_4k_6 - 4k_5^2 \tag{3.58}$$

Subtracting the $16k_4k_6$ term from both sides and dividing through by 4 yields:

$$k_4^2 + k_6^2 - 2k_4k_6 \geqslant -k_5^2 \tag{3.59}$$

This will always be true if:

$$k_4^2 - k_6^2 - 2k_4k_6 \geqslant 0 \tag{3.60}$$

which must be true because

$$k_4^2 - 2k_4k_6 + k_6^2 = (k_4 - k_6)^2 \geqslant 0 \tag{3.61}$$

In summary, due to the choice of a quadratic form for the fit to the window of voxels surrounding the voxel of interest, and based on the window of voxel's being symmetrically distributed. the maximum slope at the voxel and the eigenvalues of the voxel's Hessian matrix can be easily calculated using Equations 3.50 and 3.56.

## D. TERRAIN CELL CLASSIFICATION BASED ON QUADRATIC APPROXIMATIONS

The limited numbers of ways a two dimensional quadratic surface can behave is the basis for this algorithm that classifies terrain cells. The behavior of a quadratic surface can be completely described by the gradient (slope as derived from Equation 3.50) and the eigenvalues of the Hessian matrix (Equation 3.56). To see how this is true. the following derivation again starts at Equation 3.2, however, this time the notation is changed to comply with the notation used in Reference 30. Acknowledgement is given to Professor McGhee for allowing the use of his notes in this section of this thesis. Equation 3.2 is rewritten as:

$$z = a + b_1x + b_2y + c_{11}x^2 + c_{21}xy + c_{12}xy + c_{22}y^2 \tag{3.62}$$

and in it's matrix form as:

$$= a - b_1 x - b_2 y + \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.63}$$

or

$$= a - \vec{b}^T \vec{p} + \vec{p}^T C \vec{p} \tag{3.64}$$

where

$$\vec{p} = \begin{bmatrix} x & y \end{bmatrix}^T \tag{3.65}$$

and

$$C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \tag{3.66}$$

Without any loss of generality, the above equations can be simplified by letting $c_{12} = c_{21}$. The directional derivatives of Equation 3.62 can then be written as:

$$\frac{\partial z}{\partial x} = b_1 + 2c_{11}x + 2c_{12}y \tag{3.67}$$

and

$$\frac{\partial z}{\partial y} = b_2 + 2c_{12}x + 2c_{22}y \tag{3.68}$$

In matrix notation the gradient of z is written as:

$$\nabla z = \vec{b} + 2C\vec{p} \tag{3.69}$$

and the transpose of gradient z is:

$$\nabla z^T = \vec{b}^T - 2\vec{p}^T C \tag{3.70}$$

The four second partial directional derivatives of z are written as:

$$\frac{\partial^2 z}{\partial x^2} = 2c_{11} \tag{3.71}$$

$$\frac{\partial^2 z}{\partial y^2} = 2c_{22} \tag{3.72}$$

$$\frac{\partial^2 z}{\partial y \partial x} = \frac{\partial^2 z}{\partial x \partial y} = 2c_{12} \tag{3.73}$$

From this, it is seen that the Hessian matrix is simply:

$$H = 2C \tag{3.74}$$

Expanding z in a three-dimensional MacLaurin series about the origin, z can be expressed as:

$$z = z_0 + \nabla z^T(0)\vec{p} + \frac{1}{2}\vec{p}^T H\vec{p} \tag{3.75}$$

This is an exact expansion of the quadratic equation even though the MacLaurin series is an infinite series. This is true since the third and higher order derivatives of a quadratic equation are all identically zero and the MacLaurin series is an infinite series of increasing order derivatives.

Defining a contour line as a line connecting points of equal altitude (z), the equation of a contour line can be derived. For a given contour line $n$, let $z = n\Delta z = z_n$. Then, from Equation 3.75, the equation for contour line $n$ is thus:

$$z_0 - z_n - b^T\vec{p} - \frac{1}{2}p^T H\vec{p} = 0 \tag{3.76}$$

Note that without loss of generality. the origin of the x. y coordinate system can be chosen such that $\nabla z = 0$ (providing that such a point exists). If this is done. Equation 3.76 simplifies to

$$p^T H\vec{p} = z_0 - z_n = c_n \tag{3.77}$$

This equation can be further simplified by performing a coordinate rotation in order to eliminate any $xy$ cross product terms. This results in a diagonalization of the Hessian matrix. This is accomplished by means of a *similarity transformation.* Specifically. suppose that:

$$\vec{p} = \begin{bmatrix} x \\ y \end{bmatrix} = T\begin{bmatrix} u \\ v \end{bmatrix} = T\vec{q} \tag{3.78}$$

and T is defined to be an orthonormal (rotation) matrix. Then, evidently:

$$\vec{p}^T H\vec{p} = \vec{q}^T T^T HT\vec{q} = q^T \Lambda q \tag{3.79}$$

where the middle term is the similarity transformation and by definition:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = T^T HT = T^{-1}HT \tag{3.80}$$

Once again as described in the previous section. $\lambda_1$ and $\lambda_2$ are the eigenvalues of the Hessian matrix (H) and T is called the matrix of eigenvectors of the Hessian. The eigenvalues can be solved for, exactly as previously described. and by doing so, the equation for a contour line can now be written as:

$$\vec{p}^{\,T} H\vec{p} = \begin{bmatrix} u & v \end{bmatrix} \Lambda \begin{bmatrix} u \\ v \end{bmatrix} = \lambda_1 u^2 - \lambda_2 v^2 = c_n \tag{3.81}$$

This solution is based only on the surface being modeled by a quadratic equation, as was done in the previous section of this chapter, with an appropriate choice of coordinates to remove the Gradient term and any cross products between $x$ and $y$. The behavior of Equation (3.81) is dependent exclusively on the eigenvalues, $\lambda_1$ and $\lambda_2$ and therefore the behavior of the surface is a function of $\lambda_1$ and $\lambda_2$ only. In examining the effect of the eigenvalues, it is seen that there are three major situations that can arise. These are the case where $\lambda_1$ and $\lambda_2$ are both of the same sign, when they are of different signs, and when one or both of the eigenvalues is zero. Each of these cases is treated, in turn, in the following analysis.

### 1. Case I

Case I is where $\lambda_1$ and $\lambda_2$ are both of the same sign. Here it is sufficient to consider both eigenvalues to be positive to determine the shape of the contour lines. If $\lambda_1 = \lambda_2$ then Equation 3.81 becomes:

$$\lambda_1\left( u^2 + v^2 \right) = c_n \tag{3.82}$$

or

$$u^2 + v^2 = \frac{c_n}{\lambda_1} = r^2 \tag{3.83}$$

which is the equation of a circle. Furthermore. if the eigenvalues are not equal. $\lambda_1 \neq \lambda_2$. then Equation 3.81 becomes the equation of an ellipse. As a side note. the major axis of the ellipse has a length of:

$$l_1 = 2\left(\frac{c_n}{\lambda_1}\right)^{\frac{1}{2}} \tag{3.84}$$

and the minor axis has a length of:

$$l_2 = 2\left(\frac{c_n}{\lambda_2}\right)^{\frac{1}{2}} \tag{3.85}$$

The sign of the eigenvalues has the effect of determining whether the contour lines are growing upward or downward. If they are both negative. then. from Equation 3.81. as $u$ or $v$ expand out from the origin (get bigger), the contours. $c_n$ becomes more negative. ie.. the elevation decreases, and so the point can be considered a *hilltop.* Inversely, if both eigenvalues are positive, as $u$ and $v$ get farther from the origin, the elevation increases and the point is a *depression.*

### 2. Case II

Case II is where the two eigenvalues have opposite signs. Consider the case where the largest eigenvalue, $\lambda_1$, is positive. Then the equation of a contour line , Equation 3.81, becomes:

$$\lambda_1 u^2 - \left|\lambda_2\right| v^2 = c_n \tag{3.86}$$

This is the equation of a hyperbola. As the magnitude of the $u$ coordinate increases the elevation increases, however, as the $v$ coordinate increases the elevation decreases. Thinking in terms of a three dimensional surface, the terrain analogy to this surface is a *pass* which is a low depression through a high terrain structure. If the sign of the eigenvalues is reversed, the increase in the elevation due to the $u$ term is less significant than the decrease in elevation due to the $v$ term. To distinguish between the two situations of this case, this situation can be thought of describing a *saddle* terrain structure. The difference between these two cases is obtuse but very real and needs to be considered at length.

### 3. Case III

The third and final case arises when one or both of the eigenvalues is zero. Clearly, it is an impossible situation to have the larger eigenvalue in magnitude, $\lambda_1$, equal to zero and then to have the smaller eigenvalue in magnitude, $\lambda_2$, either negative or positive in value. Similarly, the situation when both eigenvalues are zero is equally simple to analyze in that the equation of a contour line is identically equal to zero in all directions and therefore the region is a *flat* or planar region. This is not to be confused with a level region. A flat region has little or no curvature whereas a level region has little or no slope.

Consider the situation in which the smaller eigenvalue, $\lambda_2$, is equal to zero and $\lambda_1$ is positive. Referring again to the equation of a contour line, in the $v$ direction there is no change in the elevation while in the $\pm u$ direction the elevation increases. This is a *valley* terrain element. Similarly, if $\lambda_1$, the larger eigenvalue, is negative, then the terrain remains level in the $v$ direction but decreases in the $\pm u$ direction. This is the inverse of a valley and is called a *ridge* terrain element. (As a note, since nothing in nature is ever absolutely zero, there has to be a threshold applied when determining that one or both of the eigenvalues of the surface is zero. It will be seen later in the chapter describing the results of the simulation that this threshold, appropriately called in the computer simulation "curv_threshold", has a significant effect on the performance of the simulation.)

TABLE 2

CLASSIFICATION OF SURFACE VOXELS
BY EIGENVALUES OF HESSIAN MATRIX

$\lambda_1$ = eigenvalue with larger magnitude
$\lambda_2$ = eigenvalue with smaller magnitude

| sign of $\lambda_2$ | sign of $\lambda_1$ | | |
|---|---|---|---|
| | - | 0 | + |
| + | saddle | impossible | depression |
| 0 | ridge | planar | valley |
| - | hill | impossible | pass |

The above three cases represent the only ways in which Equation 3.81 can possibly behave. By deciding to model the voxel by a quadratic, the terrain categories into which a voxel can be classified has been limited to the previously described seven terrain forms. These are summarized in TABLE 2 where the sign of the larger eigenvalue defines the columns and the value of the smaller eigenvalue defines each row. It is interesting to note that the terrain types in column one. defined by a negative $\lambda_1$. can all be considered as modified ridges. A saddle is a ridge which does not actually remain flat on top but rather has a smaller amount of upward (positive) curvature associated with it. Similarly, a hill is a ridge which does not remain level on top but curves down in all directions. Also, column three can be considered as a set of modified valley terrain types. A depression (or pit) is a valley which does not remain flat at it's bottom but rather curves upward in both directions. And, lastly, a pass can be thought of as a valley (through some other elevated terrain feature) which is associated with a small amount of downward curvature at its bottom.

### 4. The Effect of Slope

Part of the above analysis required setting the coordinate origin at a point such that the Gradient is zero. From Equation 3.50 it is seen that the Gradient, and therefore the maximum slope of the voxel, is dependent only on the constants $k_2$ and $k_3$ while the eigenvalues are based only on the constants $k_4$, $k_5$ and $k_6$. Therefore, the effect of the eigenvalues is independent of the slope and the above seven terrain classification types are not really dependent on the slope being zero. Logically. if this is true. the slope could be another way, independent of the eigenvalues, of classifying terrain. Thus, voxel's with a slope equal to zero, within some threshold value, will be called *level* terrain. Terrain with a slope of something greater than the zero threshold value but less than an unsafe slope threshold will be designated as *safe slope* terrain. Anything else is called *unsafe slope* terrain.

### 5. Generalized Classification Rules

In this simulation it was decided to use the slope information in the quadratic model as the primary classifier for the voxel. This was due primarily to two reasons. First, in trying to identify specific topographic features, the usual convention is to classify terrain based on the tops of hills and ridges or the bottoms of valleys and depressions. Small hills on the sides of larger hillsides may or may not be significant. The same can be said of small ridge lines running along or up a hill side or down a valley wall. Secondly, when considering the concept of mobility, a robotic vehicle trying to navigate through an unpreviewed terrain scene would be more interested in following flat terrain features, such as stepping from hilltop to hilltop, than in trying to follow some other specific terrain features which might be crossing some sort of steep slope. So the data is presented to highlight these level terrain features. This forces the slope to be the most significant terrain classifier. Admittedly, this is a somewhat arbitrary choice and therefore it deserves further study.

In what follows, each voxel is not only given a primary classifier, but it is also given a secondary classifier. The secondary classifier is the information derived from the eigenvalues. These seven classification categories contain the curvature information of the terrain. Not only will an autonomous vehicle have to try to utilize level terrain but it must also be able to tell the difference between specific level terrain categories. Obviously, to a legged vehicle there is a significant difference between a flat hilltop and the flat bottom of a depression. This implies that each voxel must be classified into one of 21 possible categories. Based on Equations 3.50 and 3.56. this classification is possible and can be accomplished with a reasonable degree of computation ease.

## E. VALIDATION OF THE CLASSIFICATION MODEL

Before attempting to utilize the algorithm to characterize voxels, as derived above, in a large scale simulation, the concept had to be validated. This was done in BASIC on an ATARI 520 ST personal computer. BASIC was chosen over other languages due to it's ease of use and rapid debugging capability allowing the code to be written in a minimum amount of time. Any of a number of other languages were available and could have just effectively been used.

The program used was a straightforward implementation using Equations 3.31-3.33 and Equations 3.40-3.41 to calculate the constants necessary for Equations 3.50 and 3.56. There was a second set of runs done of this validation model using the second method of deriving the coefficients. The shape of the resulting curves was exactly the same, since the only difference was that with the first method the constant $k_1$ had to be guessed at and in the second set of runs the constant was calculated. The constant $k_1$ only effects the vertical displacement of the window as a whole and not it's shape.

The input to the program was a data base of 46 different scenes which were generated deterministically. There was no attempt to make the data base comprehensive.

The methodology worked as expected with few exceptions. Most terrain features were characterized without error. The one major terrain type which was unexpectedly generally characterized in error was a ridge or valley which was oriented diagonally across the terrain scene. When oriented either port and starboard, or fore and aft, all valleys and ridges were correctly identified. However when they were diagonalized, the routine identified them as a saddle or pass terrain feature. Upon further investigation, it was discovered that the quadratic least squares fit to such a terrain scene does not preserve diagonal valleys or ridges. By calculating the elevations of the voxels by using the quadratic fit based on the calculated constants ($k_{1-6}$) and then comparing them to the data base elevations, it became apparent that the least squares fit to a diagonal valley or ridge was indeed a saddle or pass terrain feature.

The following are examples are just 5 of the 46 test terrains. In each table, the first entry will be the actual terrain type, the second will be the array of actual elevation values and the third will be the array of elevations based on the quadratic least squares fit. After that will be the calculated larger eigenvalue, $\lambda_1$, the smaller eigenvalue, $\lambda_2$, the calculated slope, in degrees, and finally the terrain type as characterized by the program. For example, in TABLE 3, the terrain is a peak with no slope. The actual elevation is the first 3x3 matrix and the calculated

TABLE 3

PEAK NO-SLOPE

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 2 | 1 |
| 1 | 1 | 1 |

| | | |
|---|---|---|
| 0.88 | 1.22 | 0.88 |
| 1.22 | 1.55 | 1.22 |
| 0.88 | 1.22 | 0.88 |

$\lambda_1$ = -.66
$\lambda_2$ = -.66
slope = 0

PEAK NO-SLOPE

TABLE 4

RIDGE NO-SLOPE

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 1 | 2 | 1 |
| 1 | 2 | 1 |

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 1 | 2 | 1 |
| 1 | 2 | 1 |

$\lambda_1$ = -2.0
$\lambda_2$ = 0.0
slope = 0

RIDGE NO-SLOPE

TABLE 5

PASS NO-SLOPE

| | | |
|---|---|---|
| 1.5 | 0.5 | 1.5 |
| 2.5 | 1.0 | 2.5 |
| 1.5 | 0.5 | 1.5 |

| | | |
|---|---|---|
| 1.55 | 0.39 | 1.55 |
| 2.39 | 1.22 | 2.39 |
| 1.55 | 0.39 | 1.55 |

| | | |
|---|---|---|
| $\lambda_1$ | = | 2.33 |
| $\lambda_2$ | = | -1.66 |
| slope | = | 0 |

PASS NO-SLOPE

TABLE 6

PLANAR SAFE-SLOPE

| | | |
|---|---|---|
| 1.0 | 1.1 | 1.2 |
| 1.0 | 1.1 | 1.2 |
| 1.0 | 1.1 | 1.2 |

| | | |
|---|---|---|
| 1.0 | 1.1 | 1.2 |
| 1.0 | 1.1 | 1.2 |
| 1.0 | 1.1 | 1.2 |

| | | |
|---|---|---|
| $\lambda_1$ | = | 0.0 |
| $\lambda_2$ | = | 0.0 |
| slope | = | 5.71 |

PLANAR SAFE-SLOPE

TABLE 7

RIDGE NO-SLOPE

| | | |
|---|---|---|
| 2 | 1 | 0.9 |
| 1 | 2 | 1 |
| 0.9 | 1 | 2 |

| | | |
|---|---|---|
| 1.84 | 1.32 | 0.74 |
| 1.32 | 1.35 | 1.32 |
| 0.74 | 1.32 | 1.84 |

| | | |
|---|---|---|
| $\lambda_1$ | = | -0.62 |
| $\lambda_2$ | = | 0.48 |
| slope | = | 0 |

SADDLE NO-SLOPE

terrain is the second 3x3 matrix. The eigenvalue with the larger magnitude, $\lambda_1$, is -.66 as is the eigenvalue with the smaller magnitude, $\lambda_2$. The slope is calculated to be 0. Finally, on the bottom, the algorithm classifies the terrain to be a peak with no slope.

## F. SUMMARY

The basis for any valid computer simulation is a detailed mathematical model. The purpose of this chapter has been to logically develop the model used in this simulation of a terrain classification scheme. First, the terrain generation methodology was discussed. A number of different techniques were presented through which the terrain the model is to classify could be generated. Any of the methods are equally as valid though they vary in their ease of use. The method used in this simulation is a variation of a Fourier series in which the sinusoids have random origins, amplitudes, and frequencies. This method was chosen since it converges rapidly to a reasonable terrain approximation and due to the author's familiarity with the general technique.

The mathematical derivation of a least squares quadratic fit to a region was presented. By use of a simplifying assumption, that the sample points about the voxel of interest are evenly spaced, it is shown that the equations defining the coefficients of the quadratic become relatively simple. These coefficients can then be used to calculate the three characteristic parameters of a quadratic which allow it to be classified: namely, the two eigenvalues of the Hessian matrix and the slope.

Next, a theory of quadratic surfaces was presented in which it was proven that the behavior of the quadratic surface can be sufficiently described by the slope of the function and the eigenvalues of the Hessian matrix to allow it to be classified. This classification was divided into a primary and secondary category based respectively on the slope and eigenvalues. The primary classification is into one of three subcategories; level, safe-slope, and unsafe-slope terrain. The secondary classification is into one of seven subcategories; peak, depression, ridge, valley, planar, pass, and saddle.

Lastly, a validation of the algorithm was described. In that validation test one major discovery was made. That was that a diagonal ridge or valley could be incorrectly identified as a pass or saddle terrain type. This is due to the presence of a crossproduct term. $xy$. which produces a sufficiently inaccurate least squares fit to the surface that the program classifies the terrain erroneously. While the classification is erroneous. it should not be considered as an error in the algorithm. It is stictly an effect of using a quadratic approximation to the surface which could be negated by utilizing a higher order approximation to the surface.

In the next chapter the computer simulation model will be discussed. The model is written in LISP and is run on an ISI graphics workstation. This choice of language and computer generated a number of difficulties in producing the simulation. However, these problems were offset by the unique capabilities of each item.

# IV. COMPUTER SIMULATION MODEL

## A. INTRODUCTION

Normally, a computer program is written to generate numerical data which will then be either interpreted by a human or used as input to some sort of control system. Few programs are written which are expected to make deductive conclusions. This is because computers were originally built exclusively for the task of processing large volumes of complicated data which the programmer would later analyze. The machine was responsible for the number crunching and the human partner was responsible for symbolic analysis. This was an efficient division of responsibilities which prescribed to each member of the team tasks for which they were best suited.

An autonomous vehicle does not have the luxury of a man-machine interaction. The program which guides an autonomous vehicle must both numerically analyze the data and also generate the symbolic conclusions from which different courses of action can be decided on. In the case of this simulation, the program is expected to calculate the slope and the two eigenvalues of each voxel and then, by an appropriate set of rules, decide the type of terrain that data represents. In this chapter, the structure of this program will be discussed and the specific rules by which the program evaluates the data will be presented. First, though, since the structure of any program is as much a function of the desired results as it is of the choice of language and machine. a discussion of LISP and the ISI Optimum V Workstations is presented.

## B. LISP AND THE ISI OPTIMUM V WORKSTATIONS

### 1. Lisp

In the case of this program, the intent was to produce a simulation which incorporated artificial intelligence techniques and also produced a graphical rather than a tabular output. In the field of artificial intelligence there are two major languages in use today. These are PROLOG and LISP. Each language has it's own particular strengths and weaknesses. The choice of LISP as the language in which this simulation would be written was based primarily on two considerations. First, LISP (LISt Processing) [Ref. 32:p. xi] is very strong in it's ability to manipulate lists as well as perform numerical analysis. Since an image can be thought of as list of pixel values, it appeared that the list handling functions in LISP would be of value. Secondly, the ultimate purpose of this program is. potentially, to be incorporated as a tool in a larger program to do path planning and navigation for the ASV (adaptive suspension vehicle) [Ref. 33:pp. 66-71]. A proposed computer upgrade of the the ASV's systems includes a high performance LISP computer for this navigation module. Therefore, LISP appeared to be the logical choice of languages in which to write this simulation. However, Franz LISP (the dialect of LISP installed on the ISI Optimum V Workstations at the Naval Postgraduate School) has no inherent graphics capability. This was a major concern with respect to the desire for a graphical output from the simulation.

As it turned out. the choice of LISP was a very fortunate one due to the extreme versatility of the language. To justify that statement it is necessary to first explain a few of the details of how LISP operates [Ref. 32:pp. 1-385]. LISP is usually executed as an interperted language and is probably best described, as a language, as a set of list manipulating functions. LISP functions are written in terms of *s-expressions* which stands for *symbolic expressions* [Ref. 32:p. 2]. An s-expression is generally some sort of command or statement which is surrounded by a set of parentheses. (Symbolic expressions are a form of a *list* where a list is defined as a sequence of objects inside a pair of parentheses.) For example, "( times 8 3 )" is a LISP s-expression which tells the lisp interperter to multiply 8 by 3. As the LISP interperter reads an s-expression it goes into an *evaluation* process where it attempts to carry out the process requested in the s-expression through an evaluation of each item in the list which makes up that s-expression. The arguments of the s-expression (the elements of the list which follow the first item) can be either an *atom* or another s-expression. An atom is something which evaluates to itself (eg.. the number 8 in contrast to another list or s-expression). If the arguments are s-expressions then LISP identifies them as such and attempts to evaluate them. Thus LISP proceeds down through an s-expression attempting to evaluate each embedded s-expression until it gets to the bottom level where all the arguments to the s-expressions are atoms. It then returns the value of the bottom s-expression up to the next lower level of s-expression so that that level can be evaluated. Progressing from the bottom level up to the top level of the s-expression, LISP evaluates each level of s-expression based on the value returned by evaluating the next lower level until it finally evaluates the top level and returns that value [Ref. 32:pp. 1-14].

LISP functions are invoked in terms of s-expressions. Referring again to the previous example of an s-expression which is a mathematical function, namely, "( times 8 3 )", LISP attempts to evaluate the s-expression by evaluating the the list. It assumes that the first item in the list is a function name. To perform the requested function, LISP then attempts to evaluate the second and subsequent s-expressions. To evaluate those items it tries to match them with similar symbols on what is called the *oblist* which stands for *object list*. If one of those symbols is not part of the oblist then LISP doesn't known what to do with it and it returns an error such as

```
Error: Unbound Variable: (symbol_name)
```

saying that the item is not *bound* to any particular item on the oblist. When a symbol is bound, it is *interned* on the oblist [Ref. 32:p. 236]. This is done by adding the symbol to the oblist and associating it with a pointer which points to the address in memory of either a routine, list, or constant representing the value of that particular symbol. Returning to the example, as the function, in the form of an s-expression. is evaluated by LISP. LISP finds the pointer on the oblist which corresponds to "times". LISP then jumps to the routine in memory at that pointer. This routine takes the next two items in the s-expression. adds them and then returns the sum as the value of that s-expression.

What makes LISP such a dynamic language is that the oblist is not a static structure. All variables as they are input, all lists which are created, all strings, etc. have to be accessible through the oblist and so the oblist is a structure which is constantly in flux. Not only can lists and atoms. for example, be added to the oblist, but so can a user defined function. since a function is actually nothing more than a list of calls to other LISP functions. In fact, most LISP programming is done by a user writting his own list of calls to the functions which perform the sequence of manipulations he requires. Then that list is interned on the oblist under the name the user wishes to call his new function. This is done through the LISP function *defun* which stands for *define function.* After that the user can then invoke this new function in exactly the same manner as any other LISP function (by writing it in an s-expression along with it's arguments and then evaluating the s-expression). (This process of defining functions is more completely described in Reference 32, Chapter 3.)

Not only can the programmer write a new function in terms of calls to other LISP functions, but a new function can also be created by calls to *foreign functions* , i.e., functions written and compiled in "Fortran", "C", or "Pascal". This capability contributes substantially to LISP's great versatility. It is implemented through a LISP function called *cfasl*. To utilize this capability. the foreign function has to be written in one of the previously mention languages as subroutine or function (eg.. without an "main" routine) and then compiled into object code. without being assembled, by invoking the compiler with appropriate options ( -c for the "Fortran" and "C" compilers in Unix) [Ref. 34:pp. 8-4 to 8-8]. When the "cfasl" function is called, LISP reads the object code. assembles it, and then puts the pointer to the function into the oblist, bound to the name the user wishes to use to call the new function. From that point on the "foreign" function appears be a LISP function. The fact that it is written in another language and can access the unique libraries of that language is totally transparent.

This capability is not well documented in most LISP reference books and there are a few peculiarities involved in it's use. First, arguments to "foreign" functions are passed as pointers. This usually causes no problem and the programmer is often unaware of the way these arguments are passed. In Fortran variables are passed as pointers anyway, so no special efforts must be taken when "cfasl"ing Fortran routines. In "C", though. there is no set convention for passing arguments. Arguments can be passed by passing either the item or by passing a pointer to the item. A problem specifically arises. in relation to the graphics routines, when attempting to pass the Bitmap Descriptor, *BMD* , which is a pointer to the space in memory containing a bitmap. LISP passes this parameter, when cfasl'ing, by passing a pointer to a memory location where the BMD, a pointer itself, is stored. A "C" program attempting to utilize the BMD passed from the LISP environment has to declare the argument it receives as a pointer. When the "C" program attempts to retrieve the BMD by accessing the data in the memory location it must also declare the new item as a pointer type variable. This forces the "C" compiler to generate a warning since, apparently, "C" compilers do not like pointers to pointers. The warning will not abort the compilation. However, the compiler will protest vehemently.

Secondly, LISP stores "characters" and "strings" of characters differently than it does an atom. Consequently, when passing a "string" or "character" as an argument to a foreign routine (or a LISP function for that matter), the programmer must be careful to not attempt to pass the atom representing those variables instead of the "string" itself. For example, if the variable "title" in LISP has been bound to the string 'NOISY, then there is a difference between the following two LISP calls to a foreign graphics function which writes a string to a bitmap:

```
(paint_character_string "NOISY")
```

and

```
(paint_character_string title)
```

The first s-expression will work since, in LISP, anything inside double quotes is a "string". However, the second will generate an error and will not work since the variable. title. is stored as an atom. To pass a variable "string" to a foreign function. the programmer can make use of the LISP function which converts atoms into "strings". In FRANZ LISP this function is "(get_pname variable_name)" [Ref. 32:p. 251].

In summary, LISP is a very versatile and rich language. For image processing, its list processing functions provide a good set of resources to utilize. For mathematical manipulations. it has a good set of tools with which to work. It can access "foreign" functions more easily than most other languages and passing parameters to these functions is direct and, for the most part, easily accomplished. Finally, though it is a slow language in it's interperted form on most "normal" machines, there are a number of new "LISP" machines which promise performance equal and possibly superior to the performance of other compiled languages on present day high speed computers.

### 2. ISI Optimum V Workstations

The computer on which it was decided to implement this program is an ISI Optimum V Workstation. This is a graphics workstation with a Motorola 68010 microprocessor (since upgraded to a 68020 microprocessor), two megabytes of available memory and which has the UNIX 4.2BSD operating system installed.

The workstations feature a high-resolution bitmapped display for graphic display. The addressable display area on the screen is 1280 pixels width by 1024 pixels height on a 19 inch tube. As implemented on the available workstations, the display is Black and White, although color workstations are also available. In order to use the graphics environment available on the workstations. a window system allowing the presentation of multiple overlapping windows has been incorporated into the 4.2 BSD UNIX operating system. Each window can be configured as one of a number of a standard terminal types with the default window type being an ANSI x3.64 terminal running the standard 4.2 BSD UNIX tty line discipline. The graphics interfaces included in these machines are SIGGRAPH Core and GKS (Graphical Kernel System) which are two of the standardized graphics interfaces available with considerable device independence [Ref .34:pp. 1-1 to 1-8].

Besides the fact that these systems were readily available for this research, the reason for deciding on their use is their outstanding graphics capabilities. Installed as libraries, which are easily accessible to the user through either Fortran and C, are a standard set of tools allowing the user to perform a wide variety of graphical manipulations [Ref. 35:p. iii]. The libraries are divided into three levels depending on the user's requirements. The upper level is the *libtools* library. This library provides to the programmer a set of easy to use routines as interfaces for graphics programs. Typically, included at this level are routines to open and close new windows, create and manipulate menus, and to change cursor patterns [Ref. 36:pp. 2-1 to 2-6]. With the exception of opening and closing windows. little use was made of this library. The next level of library is the *libvt*, which stands for library virtual terminal. Each open window is a different terminal with it's own set of operating characteristics, even though it is just one part of the display on the physical terminal (the *desktop* ). Accordingly, each window is termed a *virtual terminal.* The libvt provides the programmer with a number of primitives for window management, pointer positioning, attribute management, object painting, clipping management, region manipulation, refresh, cursor management, font management, input buffering, and output buffering [Ref. 37:p. 1-1]. With these primitives, the programmer is provided an extensive library to generate the required graphical output on a virtual terminal. At this level, though, the programmer does not have access to the actual bitmap which is being displayed on the virtual terminal. He can only invoke the specific library routines provided which will then perform the required manipulations on the bitmap which is associated with a particular window. This relieves the programmer of the responsibility of having to manipulate individual bits on a display. For this research, there existed a requirement for printed copies of the displays to be generated for inclusion in this thesis. At the libvt level, there are no routines available which will dump copies of the displays into files which can then be printed on a printer. This requires access to the actual bitmap.

The lowest level of library in the ISI graphics system is called *libbm* which stands for library bitmap. This is the library which is most extensively used in this simulation to generate and manage the graphics required for the output. Libbm is the library in which the programmer has access to the individual bitmaps. Most of the capabilities in the libvt are present in libbm with extra routines which are appropriate when working with bitmaps [Ref. 38:pp. 2-1 to

2-16]. A bitmap is a portion of memory which holds the data the video hardware uses to produce a picture on the screen. To produce a display on a screen. the video hardware turns individual pixels on by illuminating the pixel with an electron beam from the electron gun at the back of the picture tube or off by not illuminating the pixel. The beam is scanned across the picture tube in a raster scan starting from the upper left corner down to the lower right corner, row by row. The video hardware turns the electron beam on and off according to the sequence of bits in memory, an "on" bit (1) turning the beam on and an "off" bit (0) turning off the beam. This sequence of 1's and 0's in memory is called a bitmap since it is the "map" by which the electron gun, which draws the picture on the picture tube. is controlled. The mechanics by which the libbm library actually sets and resets bits in the map will not be discussed here but the reader is directed to Reference 34 for further details.

The libbm allows a programmer to allocate a block of memory in which the bitmap will be maintained and to display that bitmap in any available open window. The library also has a set of routines, similar to the libvt, which allows the programmer to perform graphics manipulations such as drawing a line from point A to point B, setting the position of the cursor pointer, setting the line style, setting the background color, and more. By use of these routines, the programmer is relieved of the responsibility of calculating which bits need to be changed to perform these actions and how these changes effect each byte in the bitmap, yet.at the libbm level, the programmer does retain the option of manipulating individual bits. The one routine which dictated the use of the libbm library instead of the libvt library is called "bmReadBit" through which the individual bits in the bitmap can be read. This allows a function written in "C" to sequentially read the entire bitmap of a display created by the LISP simulation, convert it into hexadecimal code, and then write that code to a file which a laser printer can then use to print a copy of the display. That function, called "tofile.c", is then "cfasl"ed into a LISP function which the simulation uses to produce the graphical output. Of the foreign graphical functions written for this simulation, "tofile.c" is the only one written in "C".

A number of other foreign graphical routines were written and then "cfasl"ed into LISP (again attesting to the versatility of LISP) to perform the graphics required in this simulation. These graphical routines were all written as Fortran subroutines and integer functions. These are included as Appendix B. Each routine is documented in this appendix as to how to use it, how it is invoked, what parameters must be passed, and how it must be compiled. Consequently, these matters will not be further discussed here.

## C. STRUCTURE OF THE PROGRAM

This LISP program has been written as a file which contains the constants, user defined functions, etc., which perform the required actions. It is titled as LSCAN (Laser SCANner) and it is loaded into LISP with the command "(load 'lscan)". A copy of the file is included as Appendix C. Every attempt has been made to adequately document the purpose of each function. constant. and section and to keep the code reasonably readable without burying the code in comment. The file is written in four sections. The first section "cfasl"s the graphics routines into the oblist. The actual format of the "cfasl" command is described in the introduction to this section. The second section loads the constants which are referenced by the different user defined functions. The purpose of each constant is described as it is entered. The third section, which is the largest section, is where all the user defined functions are written. As LISP loads this portion of the file, each function is sequentially interned on the oblist. Included as a header to each function is a short definition of that function's purpose. Finally, the section which would normally be called the main program is presented. It consists of four calls to the function "(def_window)" and then a call to the function "(keep_running)". Each time "(def_window)" is called it defines the parameters of a window (window width and window height in pixels, window position in terms of x and y coordinates of the upper left corner of the window in pixels relative to the position of first window. and the window title) and attaches this list of parameters as *properties* to the window name," w1" through "w4". Finally "(def_window)" appends the window name, along with it's properties. to the end of the list "windows". The function "(keep_running)" calls the function "(draw1)" and then, iteratively calls, the function "(redraw)" until the appropriate number of runs of the simulation have been completed.

The functions "(draw1)" and "(redraw)" are very similar except that "(draw1)" initializes the bitmaps on which the graphical output is drawn while "(redraw)" only has to clear each bitmap of the previous image so that the new one can be drawn. Figure 4 is a flow chart of these two function without the the initialization or bitmap clearing routines included.

The routine "(initialize)" needs to be explained because it establishes the way the windows display the graphical output. Initialize takes the information in the list "windows" and opens a window for each window on the list. The way the program is presently written, there are four windows open. The first window is "w1" and it is the reference for positioning the other three windows. It is titled "TERRAIN" and its upper left corner is at the coordinates stored in the constants "w1x" and "w1y". The second window, "w2", is titled "NOISY TERRAIN" and it is directly below "w1". The upper left corner of "w2" is directly in line with "w1x" but displaced down by the sum of the window height and the desired vertical separation between windows (the formula in the second call to "(def_window)" ). These constants are stored as "wwheight" and "ysep" respectively. The position of windows "w3" and "w4" are to the right of "w1" and "w2" respectively. They are titled "CLASSIFICATION MAP" and "NOISY CLASSIFICATION MAP". By not writing "(initialize)" with four set of constants, but rather having "(initialize)" get the position of the four windows from the list "windows", the symbolic output of the simulation can be easily modified. New window width's, window height's, and vertical and lateral window

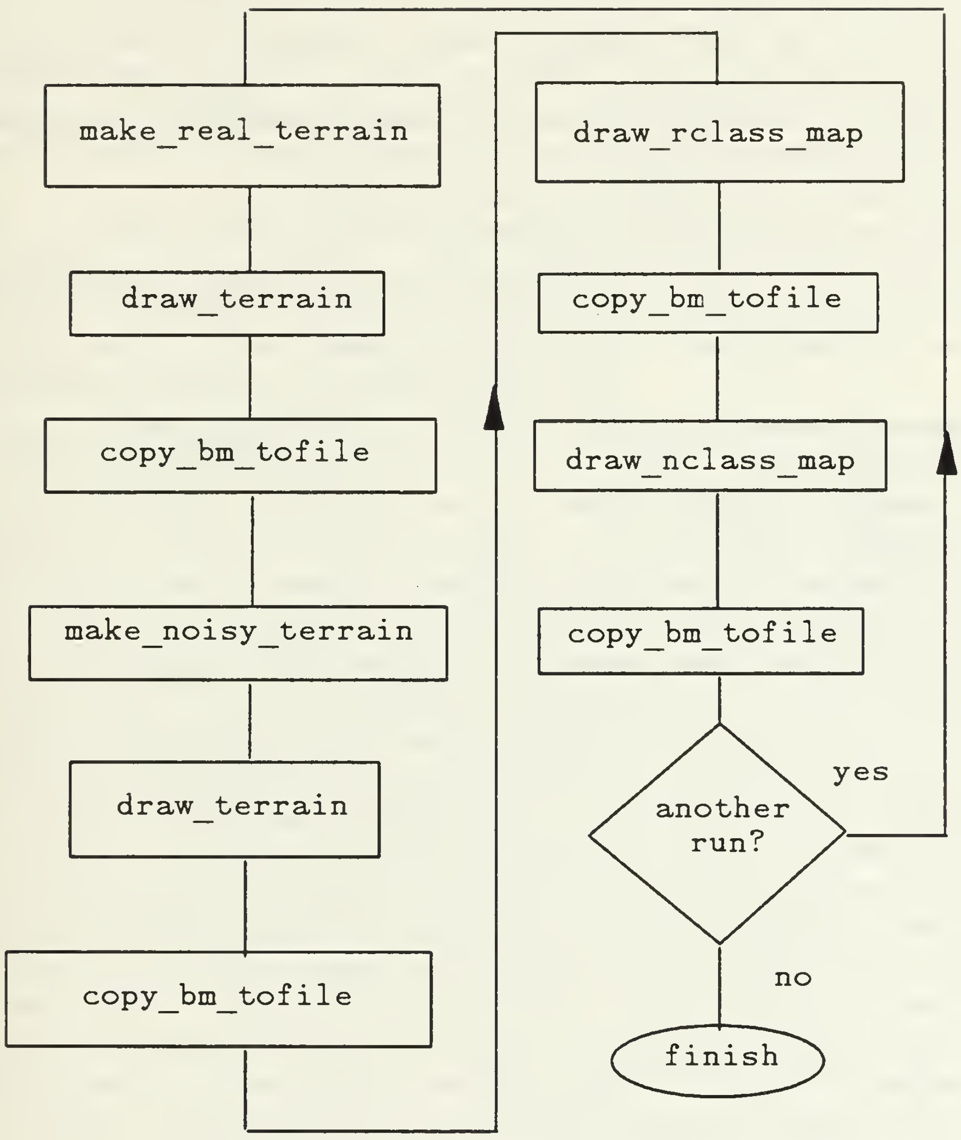

Figure 4.

Drawl and Redraw Flowchart

separation can be changed by simply changing the constants wwidth. wheight. w1x. w1y. xsep. ysep. or by changing the formula's by which the four calls to (def_window). in the "main program" section. calculate the x and y coordinates of each window. This allows the user the maximum flexibility in controlling the displays through easily modified constants and routines.

The function "(draw_terrain)" is just one of the many functions which are exclusively concerned with the graphical display of the data. They are straightforward consisting primarily of calls to set the cursor position and to draw a line to another coordinate. These functions are well documented and therefore will not be discussed further except as follows. Included in the graphics functions are a set of functions whose purpose it is to draw (paint) characters on a bitmap through a series of calls to a function to draw the required set of straight lines. The simulation uses these characters to symbolically identify each terrain voxel on the map. There are ten of these functions which effectively draw an ordinary ascii character to the bitmap. These functions were written before the anomaly about passing strings and characters to foreign functions was discovered. There was another lisp function written before those ten functions were written whose purpose it was to paint a character to the bitmap by accessing a function in libbm. "bmPaintChar". That lisp function never worked because an atom instead of a character was being passed. These functions could now easily be replaced by the other function. However, this was not done in this thesis.

The function "(copy_bm_tofile)" utilizes the foreign function. "to_file". discussed in the previous section to convert and copy the bitmap to a file that the laser printer can print. The only new thing "(copy_bm_tofile)" does is to increment a counter (named filenum) and concatenate the counter to the string "figure.___". This creates a new file name into which the function will put the bitmap. By concatenating filenum and "figure.___" together, they are converted into an atom and must be reconverted into a string.

The function "(make_real_terrain)" is where the artificial terrain is generated. The function "(make_real_terrain)" uses the random number generator to produce five random numbers. The random numbers are r1 and r2 which are the x and y coordinates of the origin of the sinusoids. r3 which determines the frequency of the sinusoid, r4 which is the amplitude of the sinusoids. and r5 which is the number of alternating sines and cosines that will be summed together to produce the elevation map. The summation of these sinusoids is stored in the array "real_terrain". The function "(make_noisy_terrain)" simply adds a random number between $\pm$ noise_limit, a constant defined in the first section of the program, to the elevations in the array "real_terrain" and then stores these new elevations into the array "noisy_terrain" to impress a random noise signal on top of the terrain.

The terrain classification is done within the functions "(draw_rclass_map)", meaning draw real (no noise) symbolic terrain map, and "(draw_nclass_map)", meaning draw noisy symbolic terrain map. by calls to the

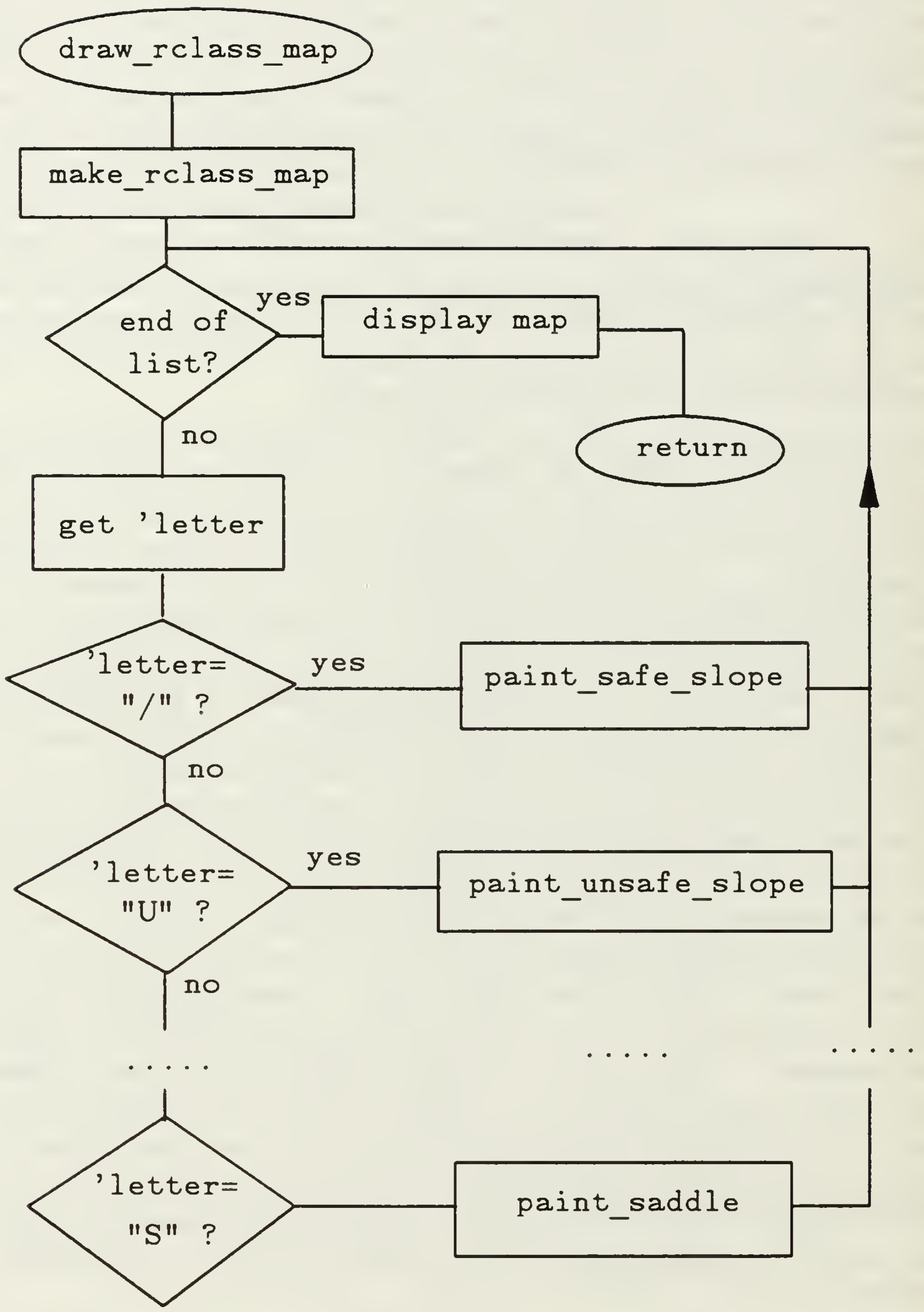

Figure 5.
Draw_rclass_map Flow Chart

functions "(make_rclass_map)" and "(make_nclass_map)". Figure 5 presents a flow chart of "(draw_rclass_map)". Since each of the functions operate in exactly the same manner, only the function "(draw_rclass_map)" is flowcharted and discussed in this chapter. To analyze the function "(draw_nclass_map)", Figure 5 is applicable with the exception that the name "rclass" is replaced by the name "nclass". The function "(draw_rclass_map)" first calls the function "(make_rclass_map)" illustrated by Figure 6. This function returns a list called *'rclass_map* in which data on each voxel is stored as a set of properties. These properties are attached to one of 900 atoms which make up the list 'rclass_map. These atoms are named 'pixel-1 through 'pixel-900. (There are 900 members to this list because the terrain scene that is being classified is 30 X 30 voxels in size.) One of the properties attached to each pixelname is a property called 'letter. This defines the symbol that will be drawn on the symbolic map. The function "(draw_rclass_map)" sequentially checks each member of the list 'rclass_map for it's 'letter and then draws that symbol onto the bitmap at the position associated with that voxel. Once the entire list has been checked and all the symbols drawn on the bitmap, "(draw_rclass_map)" displays the bitmap in the designated window on the screen.

The work of classifying the terrain voxels is done in "(make_rclass_map)" and "(make_nclass_map)". In these functions, the arrays in which the terrain maps are stored are read and the x, y, and z coordinates of each voxel are attached, as properties, to the voxel name, pixel-1 through pixel-900. As each

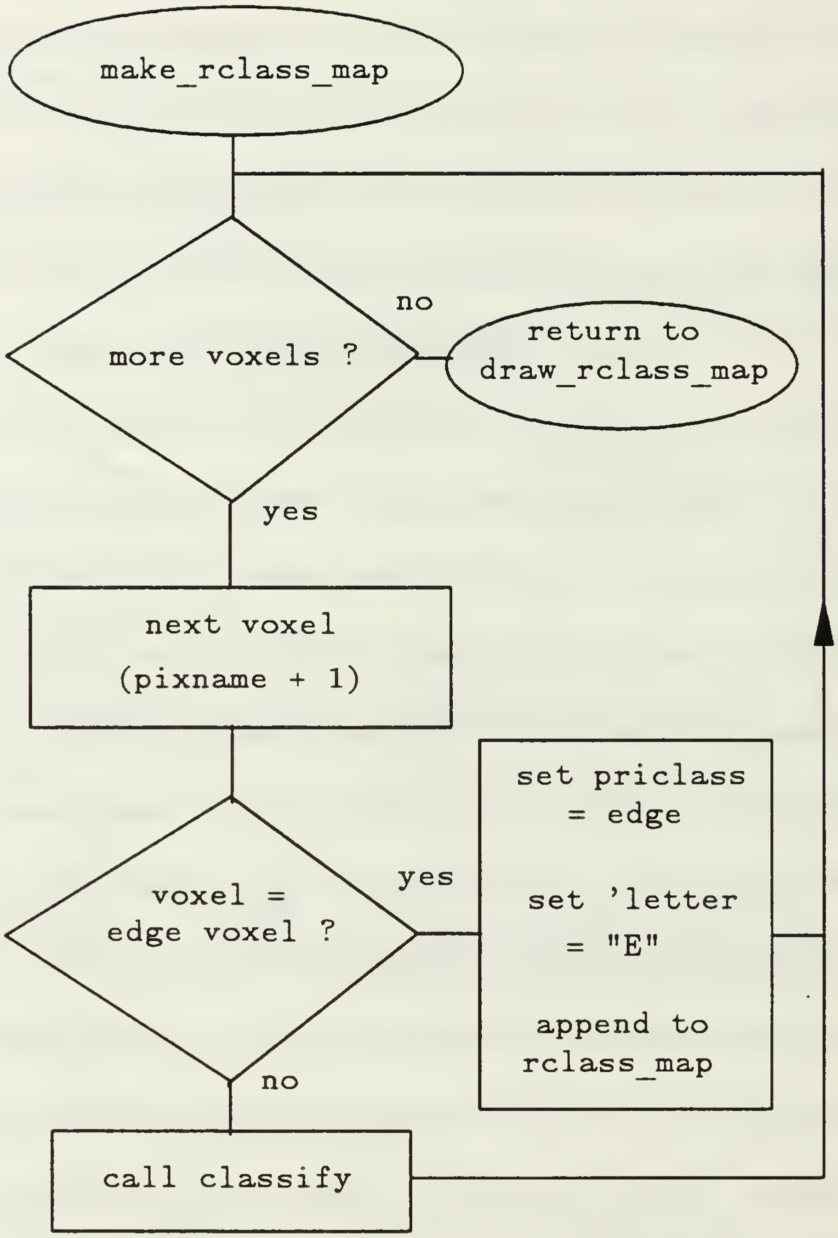

Figure 6.
make_rclass_map Flow Chart

array is read. "(make_rclass_terrain)" checks to see if the voxel is an edge voxel. which is unclassifiable (since it does not have a complete set of nearest neighbors available). If it is. then it assigns it a letter of "E". for edge. and "(make_rclass_map)" continues on to the next sequential voxel in the array. If the voxel is not an edge voxel. "(make_rclass_map)" invokes the function "(classify)" and passes to it the array name. "real_terrain", and the voxel's coordinates in the array, pi pj. The function "(classify)" then calculates the slope and the two eigenvalues for that voxel, from the previously derived formulas, and by using the rules developed in the last chapter. classifies the voxel. When it classifies the voxel. "(classify)" attaches to the voxel name the slope. the two eigenvalues. a primary classifier. a secondary classifier. and a the symbolic map. It is this 'letter which the function (draw_rclass_map) accesses to decide what to paint to the map.

After "(make_rclass_map)" and "(make_nclass_map)" have completely classified and drawn their respective symbolic maps to "w3" and "w4", "(keep_running)" checks to see if the required number of runs of the simulation have been executed. If they have, the function calls "(finish)" which closes the four windows, deallocates the space in memory reserved for the four bitmaps. and then exits to the LISP environment. If there are more runs, "(keep_running)" invokes "(redraw)" which clears the old bitmaps and then starts over again making a new set of real and noisy terrain.

## D. CLASSIFICATION AND SYMBOLS

The methodology for classifying individual pixels was discussed in the previous chapter. To reiterate, the primary classifier of a voxel is it's slope. The slope can either be level. safe slope. or unsafe slope. The thresholding between each regime is at the users discretion but, as will be shown in the next chapter, the thresholds have a major effect as to the usability of the output. The secondary classifier is based on the value of the two eigenvalues of the Hessian matrix, the larger magnitude eigenvalue being $\lambda_1$ and the smaller being $\lambda_2$. In the case of the secondary classifier. there are seven possible classes and one threshold. The threshold is the limit between a zero eigenvalue and a nonzero eigenvalue and, in the simulation, it is called "curv_threshold". This leads to a possibility of 21 different classes into which each voxel can be put. If a different symbol were to be assigned to each type class the display might be difficult for a human to interpret. Consequently, a scheme had to be developed which would reduce the number of symbols and yet preserve the quantity of information displayed. (The ASV as well as any other computer based navigation system could certainly use a more complex classification system. Limiting the classification system in favor of the display significantly reduces the apparent range of information which has been mathematically derived about the scene. For example. a line of valley pixels oriented up and down a slope could be interpreted as a gully whereas the same line oriented across a slope would be a trail or road. The differentiation between these two terrain types is obviously important for an autonomous vehicle. The generation of this kind of information and its display is a subject for further study.)

In deciding on the scheme for assigning the symbols to the classes of terrain, two primary consideration were taken into account. The first was mobility. As has been stated previously, it is felt that an autonomous vehicle should be primarily concerned with identifying level terrain for it's mobility. In general, level terrain requires the least expenditure of energy to traverse and therefore is more desirable in terms of an energy cost function.

Secondly, an autonomous system will not be able to navigate adequately if it's knowledge of it's environment is strictly in terms of single voxels. The voxels need to be aggregated into large scale terrain features in order for an autonomous vehicle to perform path planning. The knowledge of where the top of a hill is is not as good as knowing the lateral extent of that hill.

Most major terrain features seem to originate at a level point, a point with zero or near zero slope. This might be a hilltop, the bottom of a depression, the floor of a valley, or the top of a ridge line, for example. Therefore, an aggregation process would probably use points with essentially zero slope as seeds from which to "grow" major terrain feature. A peak voxel, with zero slope, which is surrounded by safe or unsafe slope voxels would aggregate into a hill.

In writting this simulation, major emphasis was given to voxels which have approximately zero slope, that was *level* voxels. It was felt that the maximum amount of knowledge possible about these voxels should be displayed even if it is at the expense of the level of the displayed knowledge about the safe and unsafe slope voxels. Accordingly, a symbol that can be somewhat associated with each of the secondary terrain types, is assigned to each voxel whose primary type is level. All other voxels which are classified as safe slope voxels are assigned a symbol of "/" and unsafe slope voxels are assigned the symbol "U". TABLE 8 is a table of the possible primary and secondary classifiers and their associated symbols.

TABLE 8.

TERRAIN SYMBOLS BY
VOXEL CLASSIFICATION

| Secondary Classifier | Primary Classifier | | |
|---|---|---|---|
| | Level | Safe Slope | Unsafe Slope |
| Flat (Planar) | - | / | U |
| Ridge | ^ | / | U |
| Saddle | S | / | U |
| Peak | (up arrow) | / | U |
| Valley | v | / | U |
| Depression | (down arrow) | / | U |
| Pass | P | / | U |

Due to difficulties in producing the symbols for the chart, a peak is presented as an up arrow on the symbolic map, while a depression is displayed as a down arrow, and a ridge and valley will be an upside down and right side up "v" respectively. In conclusion, TABLE 8 can be thought of as a kind of legend for reading the symbolic terrain maps in the next section.

## E. SUMMARY

In this chapter. the reasons for selecting LISP as the language in which to write this simulation were discussed. LISP is a rich and versatile language having capabilities most other languages do not have. One of its primary strengths besides its list processing functions is it's ability to absorb foreign functions. It was exactly this capability which allowed the program to generate its results in pictorial form instead of as reams of numeric data. In this vein, a short discussion was also presented to acquaint the reader with the graphics capabilities of the ISI Optimum V Workstations on which the simulation was run.

The program itself was also discussed. One of the other strengths of LISP is that the programmer is able to, indeed expected to, customize the language to his needs. The entire main body of the program consists of four calls to one function followed by one call the another. This is possible because of the unique structure of LISP wherein the programmer can create functions at will and intern them on the oblist so that they may be accessed just as any other LISP function.

Finally, the reasons for selecting the symbols with which to display the information produced were presented. For this simulation, it was felt that the level terrain contained the most significant knowledge and so the displays are biased in that direction. The safe and unsafe slope terrain is displayed, but other terrain characteristics have to be accessed by referencing the list of voxels named "rclass_map" and "nclass_map".

In the next chapter, the results of the simulation will be presented. Each artificial terrain sample that was created was analyzed a number of times under both noiseless and noisy conditions. Also, for each different scene, the two thresholds which effect the results, the slope and curvature thresholds, were changed and then the scene was analyzed again. The results are both interesting and surprising and show the value of this technique of terrain analysis for autonomous vehicles.

# V. EXPERIMENTAL RESULTS

## A. INTRODUCTION

In Chapter 3, a mathematical algorithm was developed to describe the behavior of a surface as modeled by a quadratic least squares fit to a 3x3 window about a point. The mathematical model was validated for a 3x3 window, a case which implies only individual point size terrain features. Except for very large scale images, most terrain features will not be limited in scope to individual voxels. For the local navigation problem, in particular, the scope of an image will be such that significant terrain features will span over a number of pixels. Therefore it is important to test the algorithm against a number of different artificial scenes to see exactly how well it can identify different major terrain features. For this test, the algorithm still fits a surface to only a 3x3 window, but it does it to each voxel in the scene. (Along the edge, the voxels do not have enough neighbors for the algorithm to work so they are immediately classified as "edge" voxels.) Individual terrain features should stand out if the method is to be of value to an autonomous vehicle.

In this chapter, the algorithm is tested on 5 different random terrain scenes. Each scene is a square 30 voxels by 30 voxels in size. These scenes range from a very benign scene with two major features to very complex scenes with a large number of individually identifiable features. Each scene was analyzed six different times, with one or the other of the thresholds which effect the performance of the algorithm, curv_threshold and slope_limit, being changed. Lastly, each sequence of tests was performed against a scene with no noise and then against the same scene with a random noise signal impressed on top of the terrain signal. The random noise signal is a random integer in [ −1, 0 , 1 ]. Each random number was multiplied by 0.1 to produce a noise level of ±0.1 distance units, where 1 distance unit is the distance between two voxels since the algorithm has been non-dimensionalized.

The symbolic results of the 5 runs are presented as Figure 7. a-n through Figure 11. a-n. The first two figures of each run are the noiseless random terrain and the noisy random terrain. The next 10 figures are the noiseless and noisy scenes as analyzed by the algorithm with the curv_threshold being reduced each time. The last two figures of each run are the analyzed noiseless and noisy scenes in which the slope_limited threshold has been increased. In all the runs the threshold of separation between safe slope and unsafe slope has been left at $30^o$. The choice of $30^o$ was purely arbitrary and not based on the operating characteristics of any vehicle in particular.

## B. GENERAL ANALYSIS

If the assumption is made that an autonomous vehicle utilizing this algorithm is capable of traversing over terrain which is classified as either level or safe slope, then this algorithm adequately analyzes each scene for the local

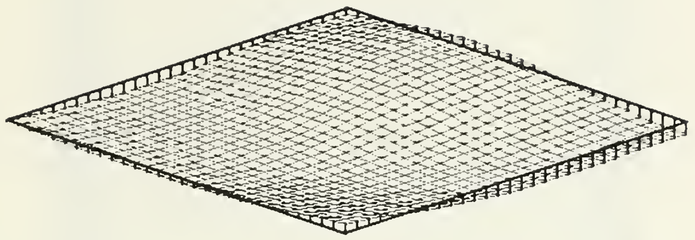

Figure 7. a. Run 1
Clean Terrain

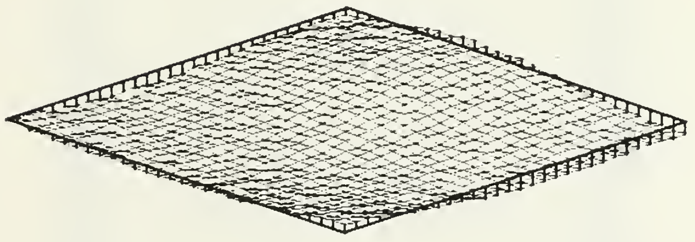

Figure 7. b. Run 1
Noisy Terrain

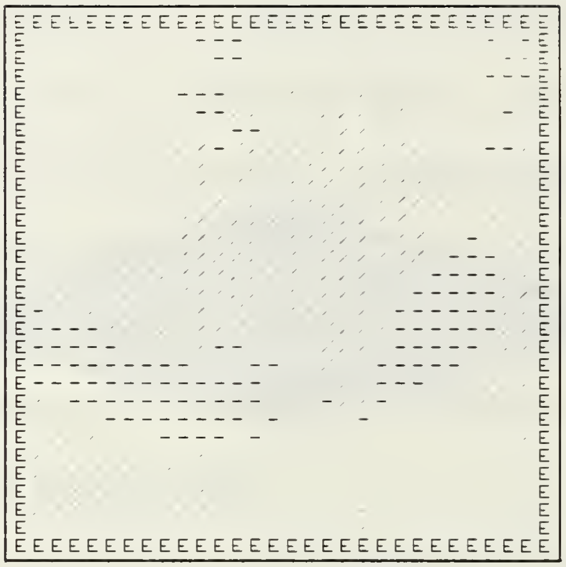

Figure 7. c. Clean Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

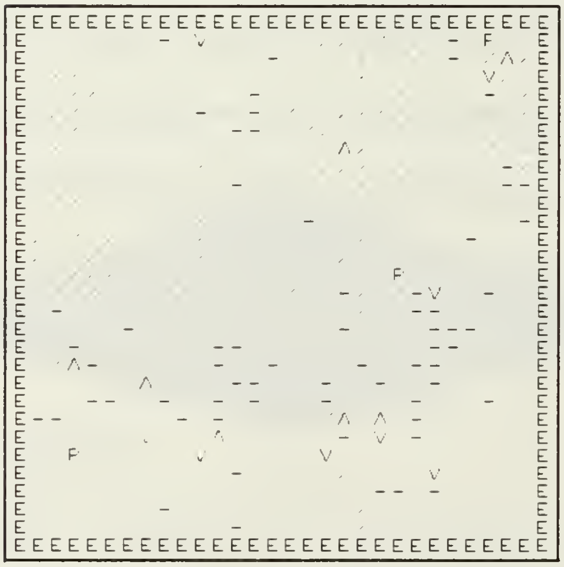

Figure 7. d. Noisy Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

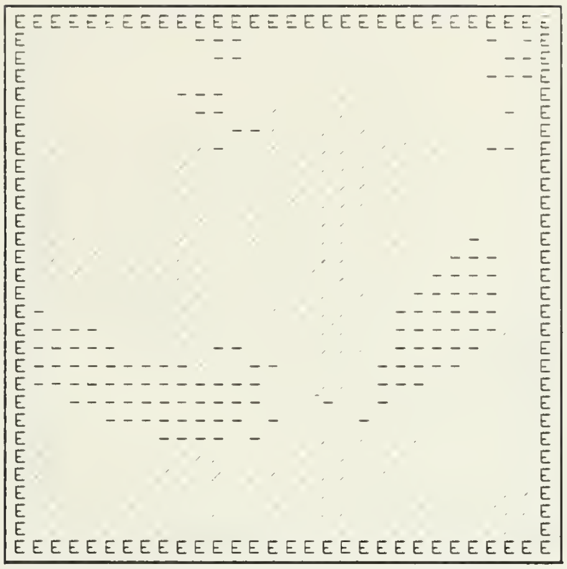

Figure 7. e. Clean Terrain

Curv_threshold = 0.15 Slope_limit = $2^o$

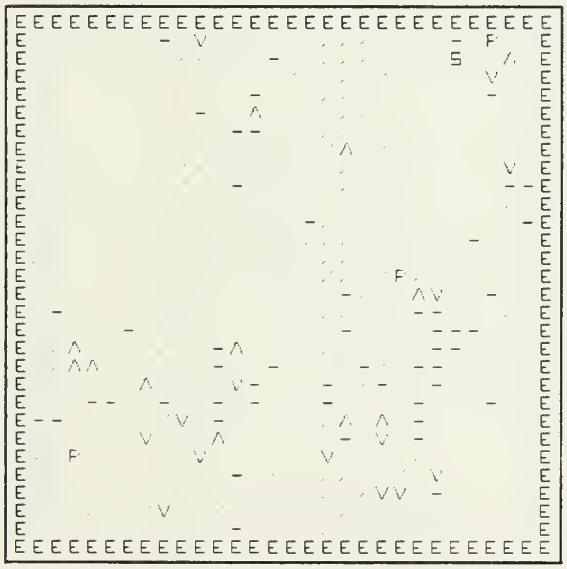

Figure 7. f. Noisy Terrain

Curv_threshold = 0.15 Slope_limit = $2^o$

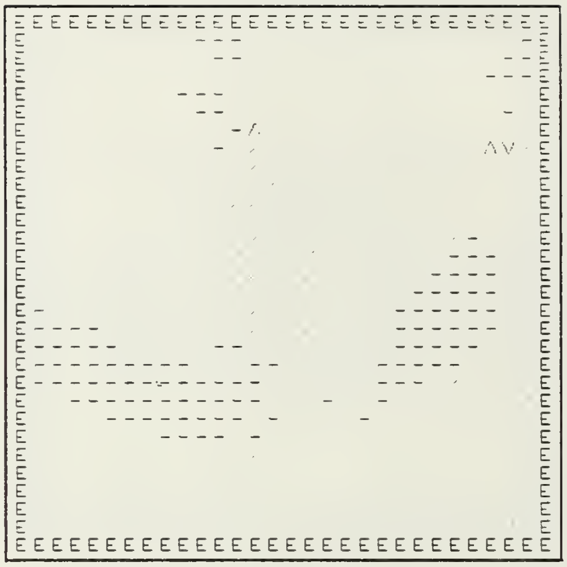
Figure 7. g. Clean Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

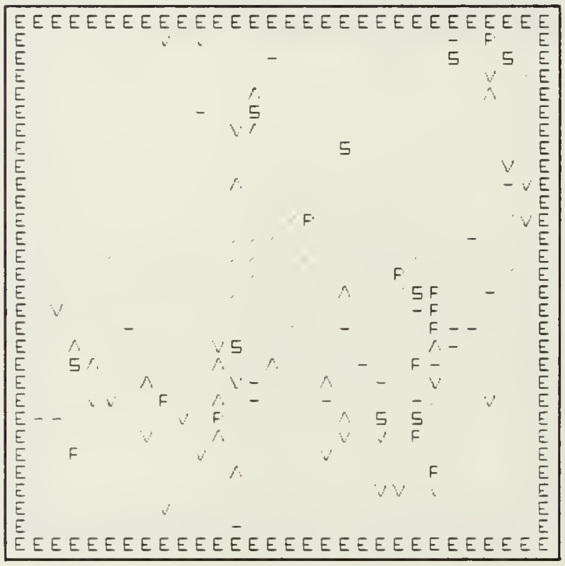
Figure 7. h. Noisy Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

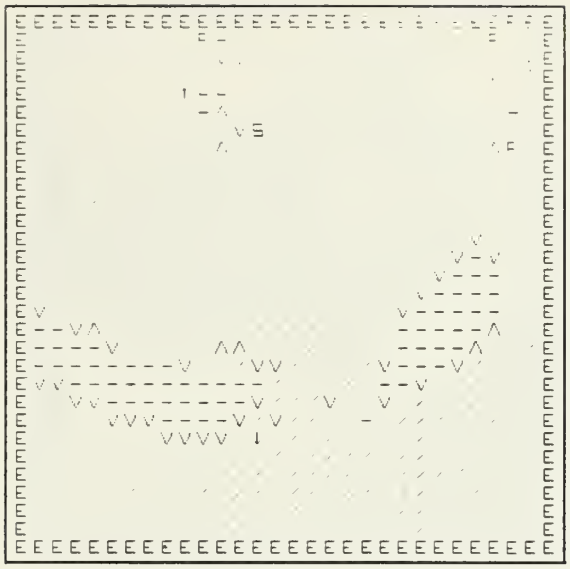

Figure 7. i. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

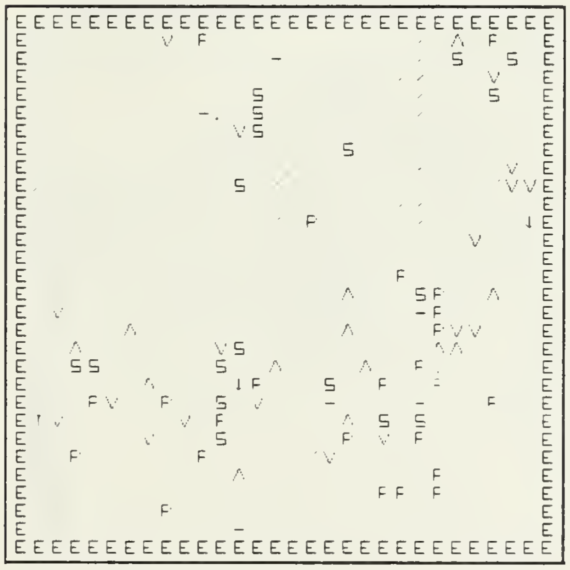

Figure 7. j. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

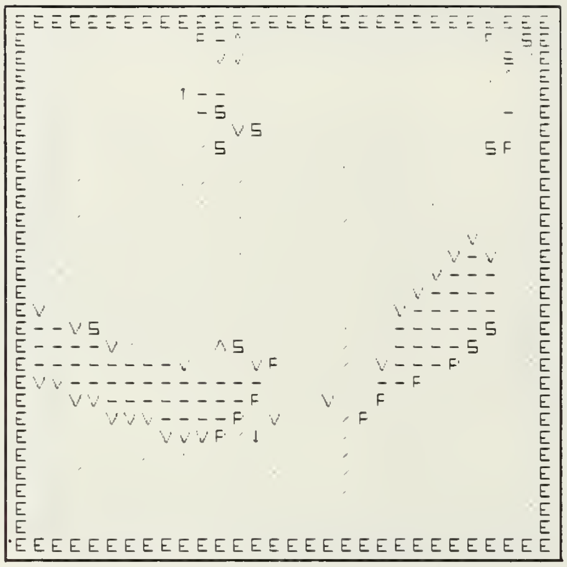

Figure 7. k. Clean Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

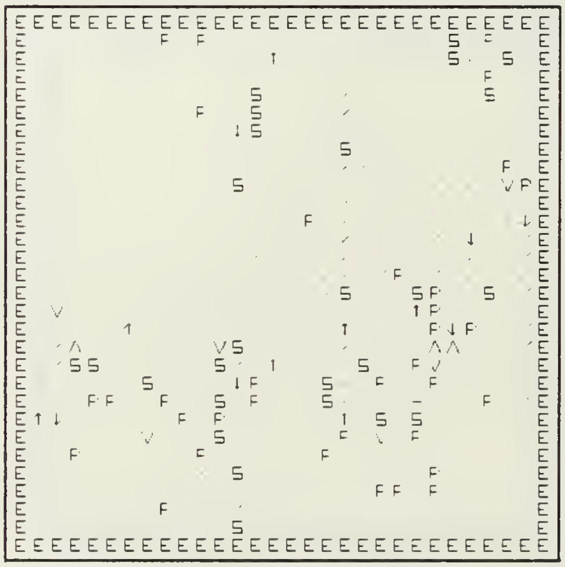

Figure 7. l. Noisy Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

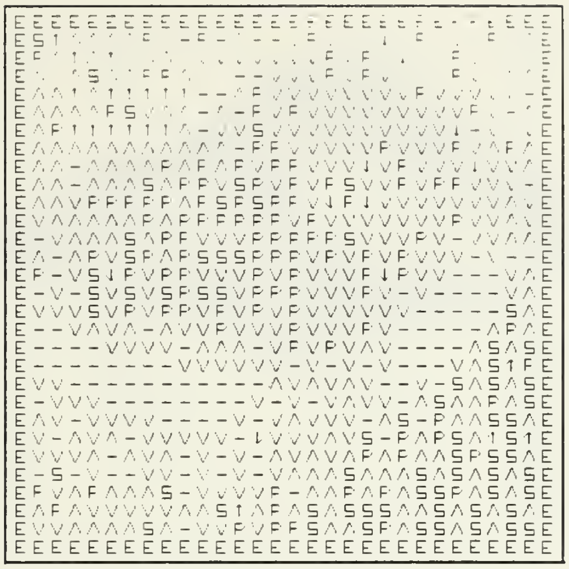

Figure 7. m. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

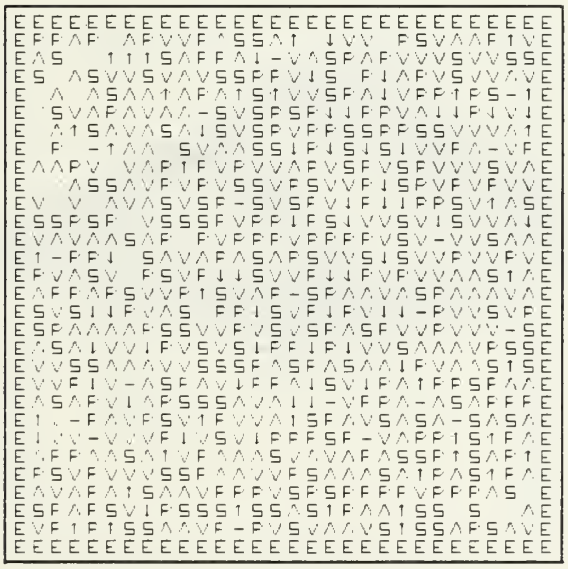

Figure 7. n. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

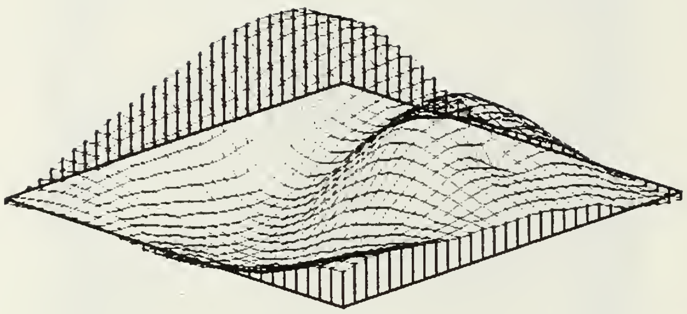

Figure 8. a. Run 2
Clean Terrain

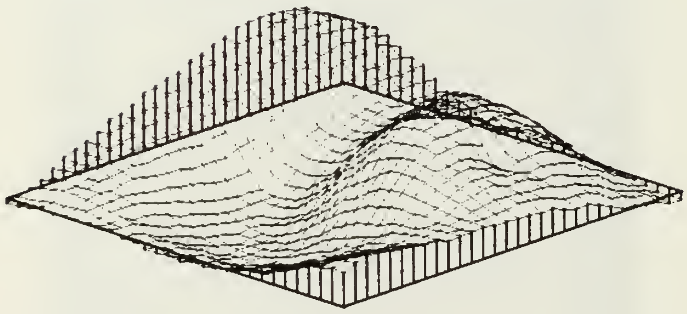

Figure 8. b. Run 2
Noisy Terrain

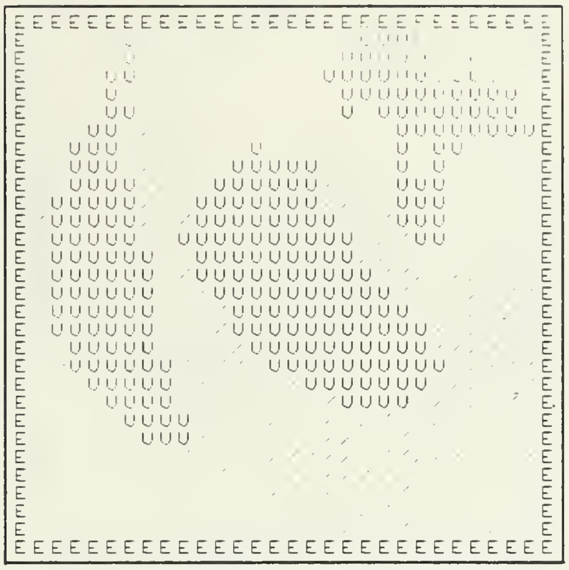

Figure 8. c. Clean Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

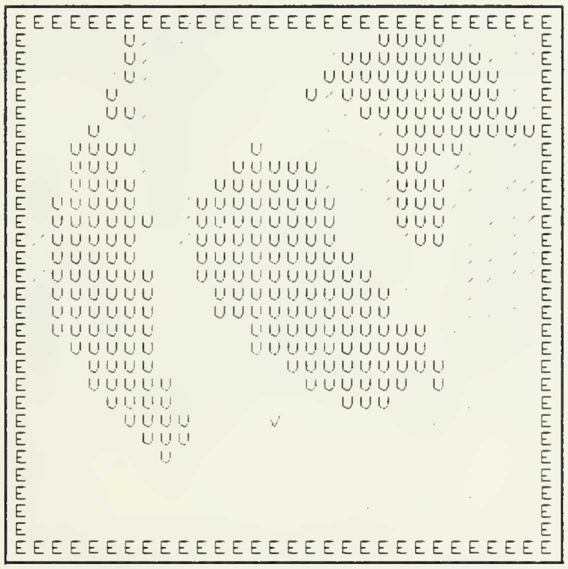

Figure 8. d. Noisy Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

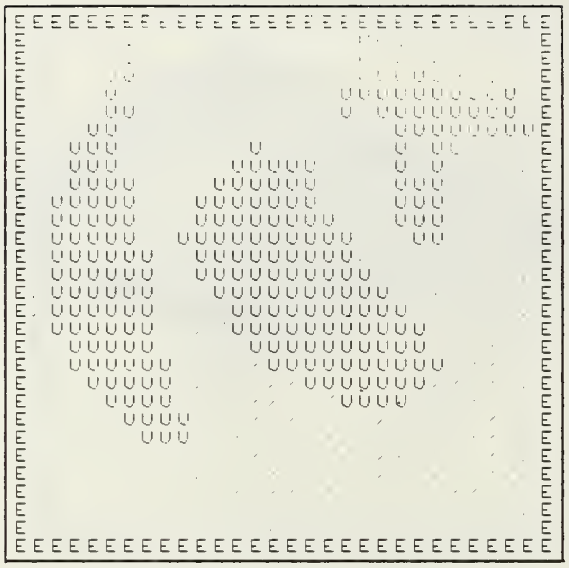

Figure 8. e. Clean Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

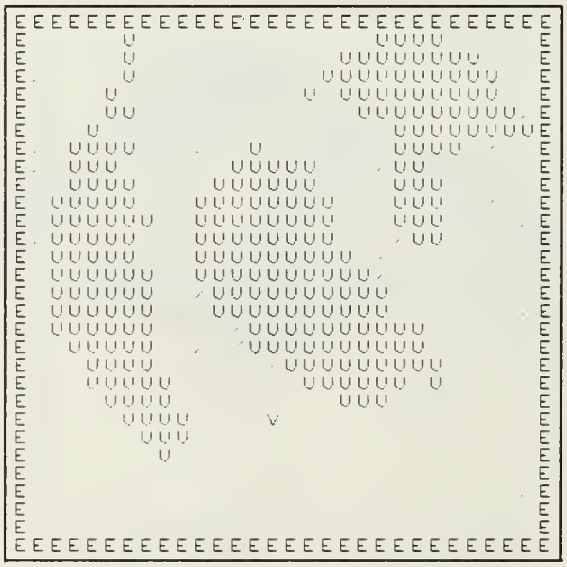

Figure 8. f. Noisy Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

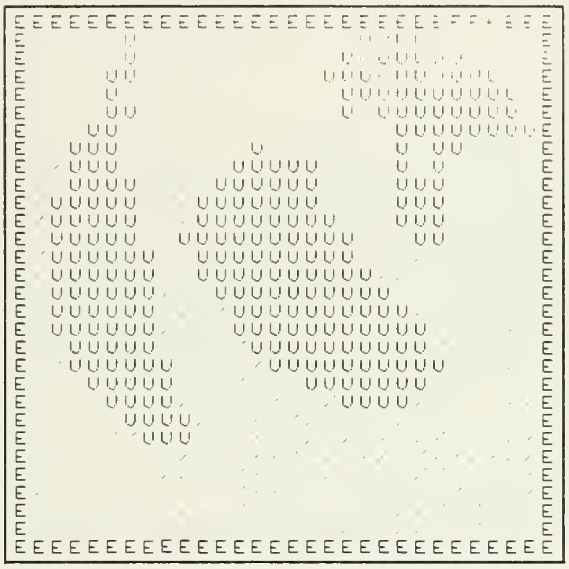

Figure 8. g. Clean Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

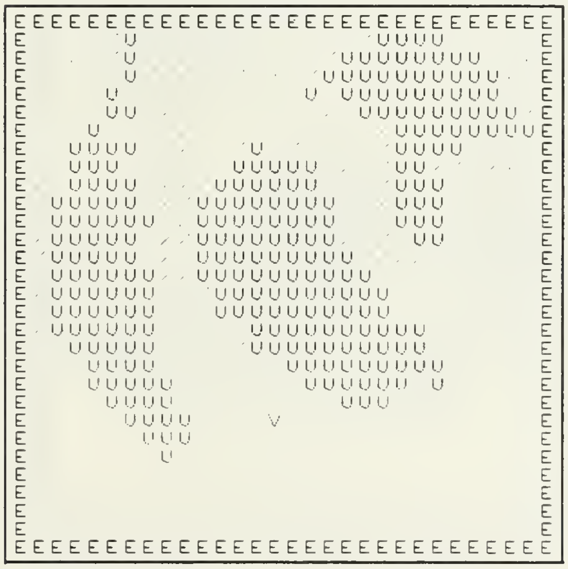

Figure 8. h. Noisy Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

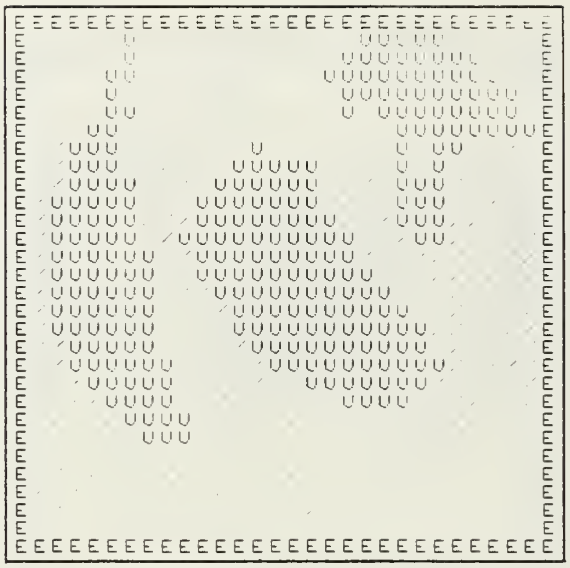

Figure 8. i. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

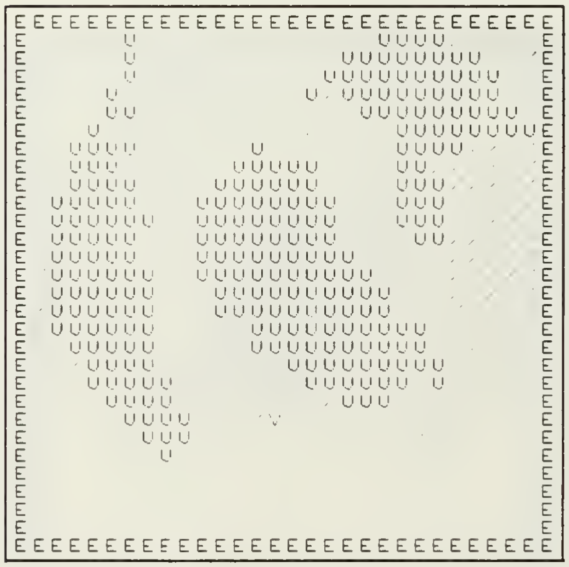

Figure 8. j. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

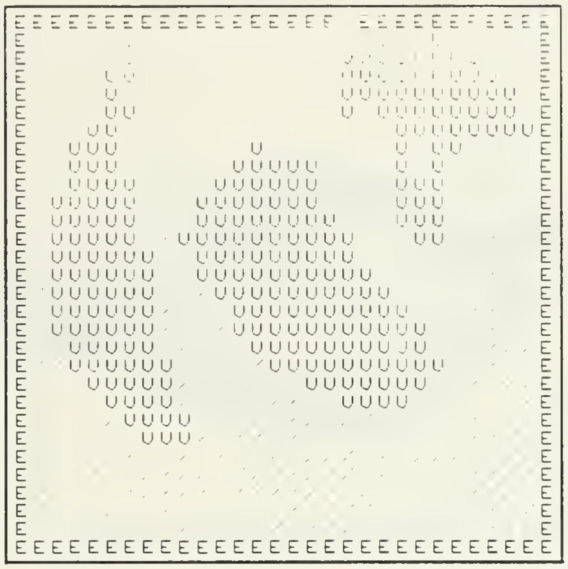

Figure 8. k. Clean Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

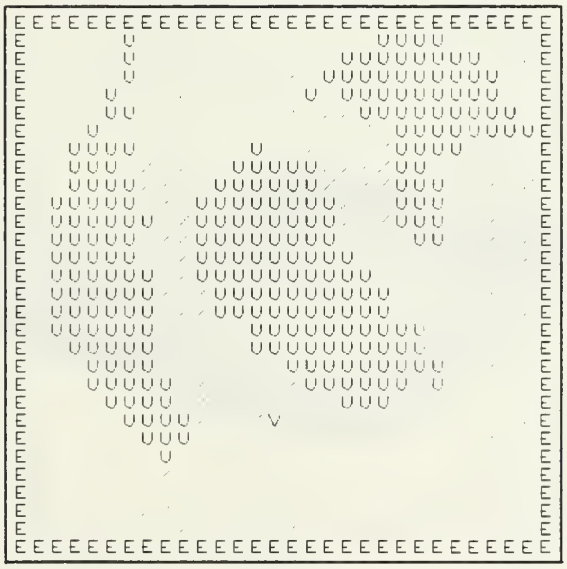

Figure 8. l. Noisy Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

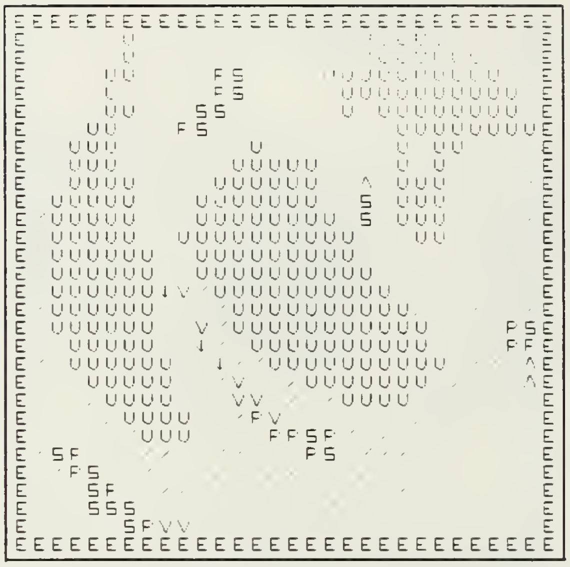

Figure 8. m. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

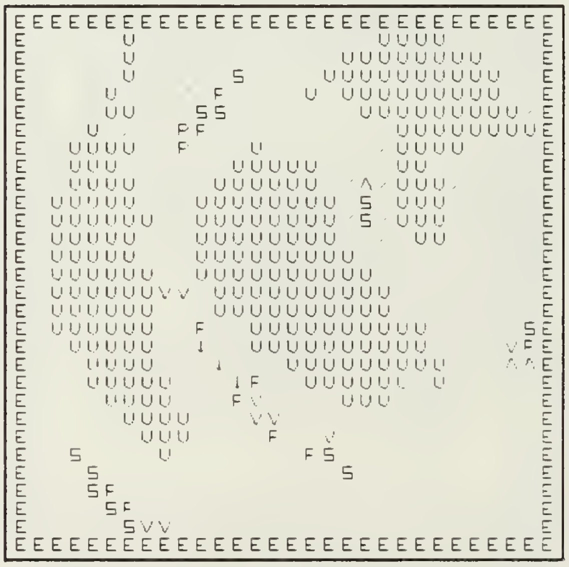

Figure 8. n. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

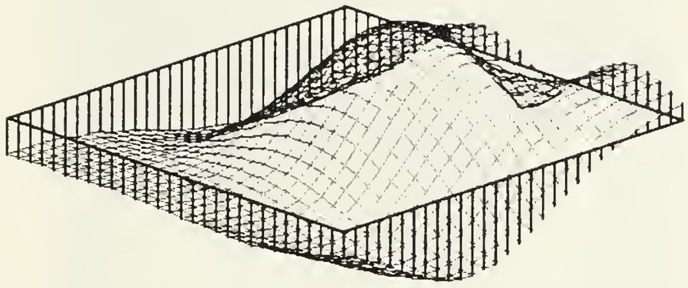

Figure 9. a. Run 3
Clean Terrain

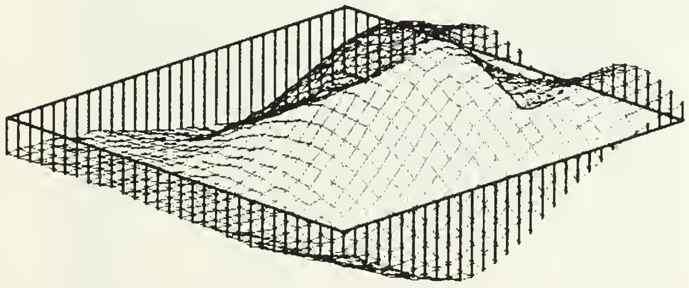

Figure 9. b. Run 3
Noisy Terrain

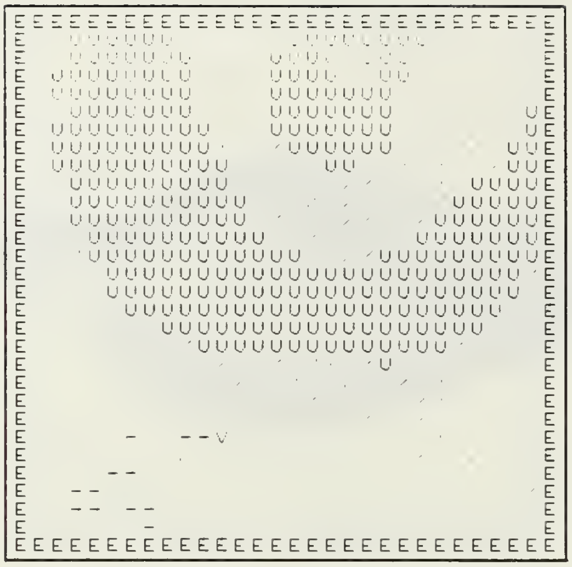

Figure 9. c. Clean Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

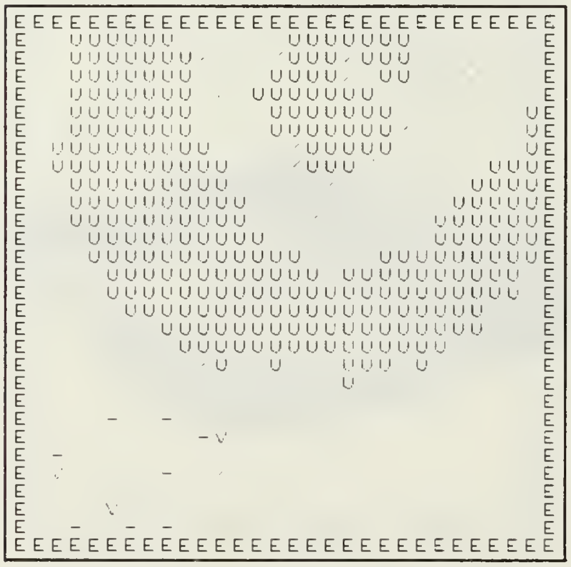

Figure 9. d. Noisy Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

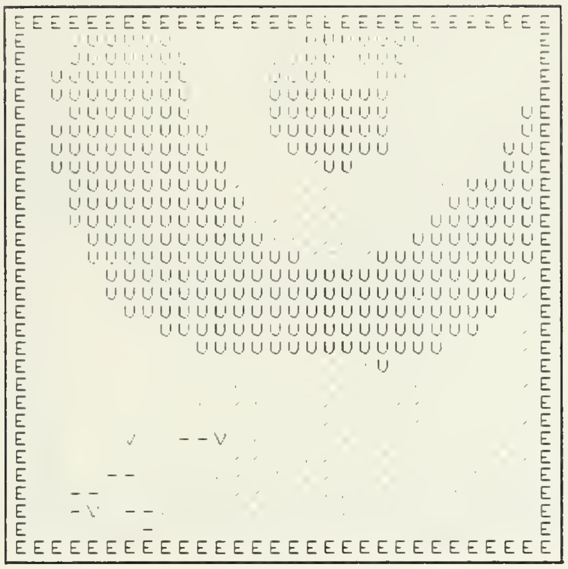

Figure 9. e. Clean Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

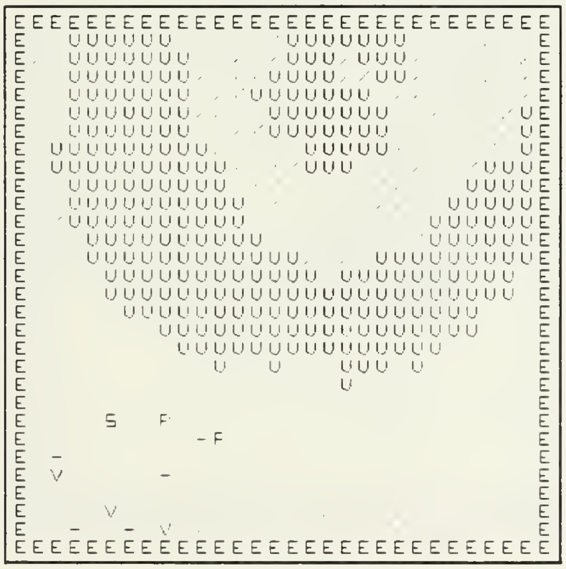

Figure 9. f. Noisy Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

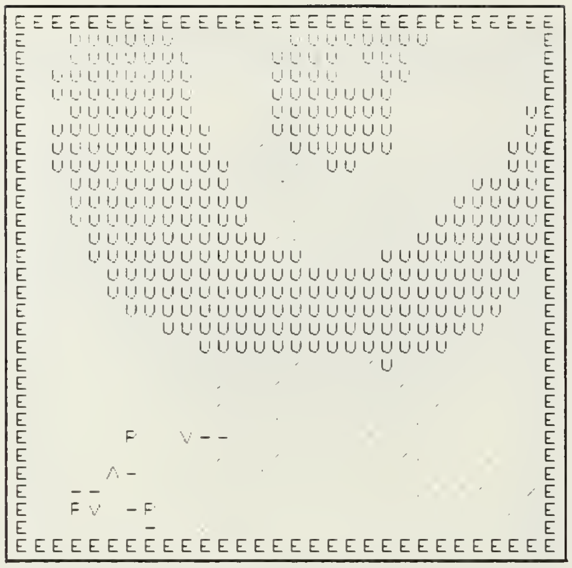

Figure 9. g. Clean Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

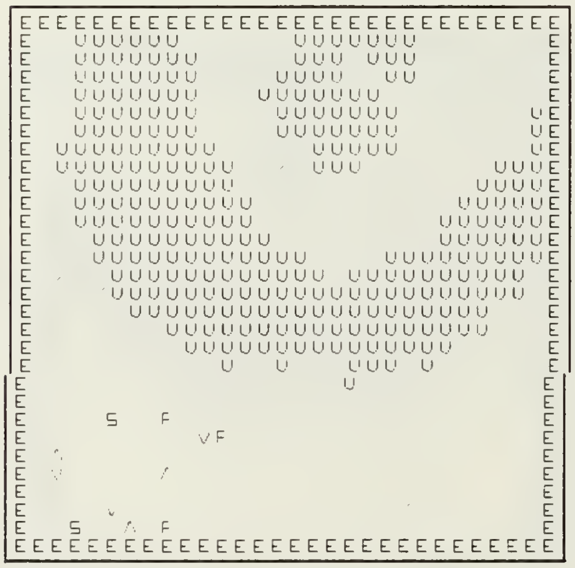

Figure 9. h. Noisy Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

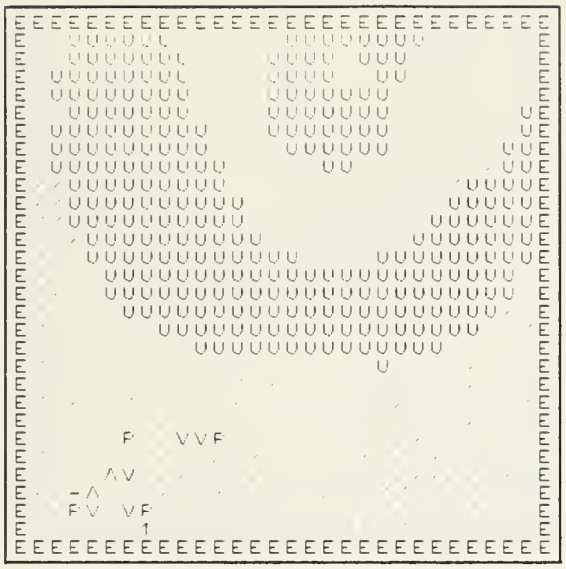

Figure 9. i. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

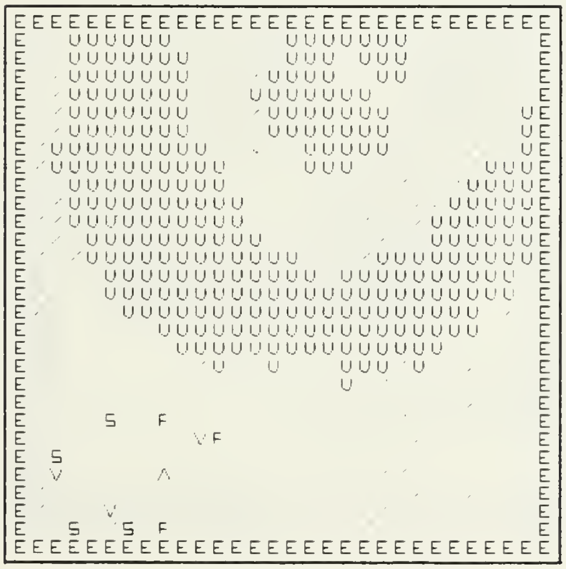

Figure 9. j. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

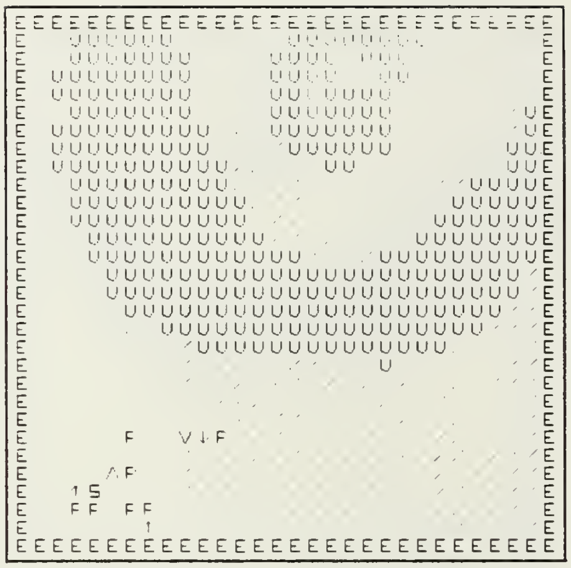

Figure 9. k. Clean Terrain
Curv_threshold = 0.01 Slope_limit = $2^{o}$

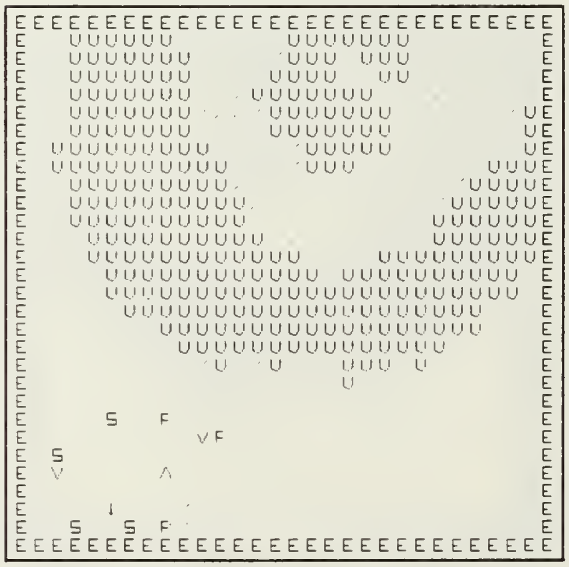

Figure 9. l. Noisy Terrain
Curv_threshold = 0.01 Slope_limit = $2^{o}$

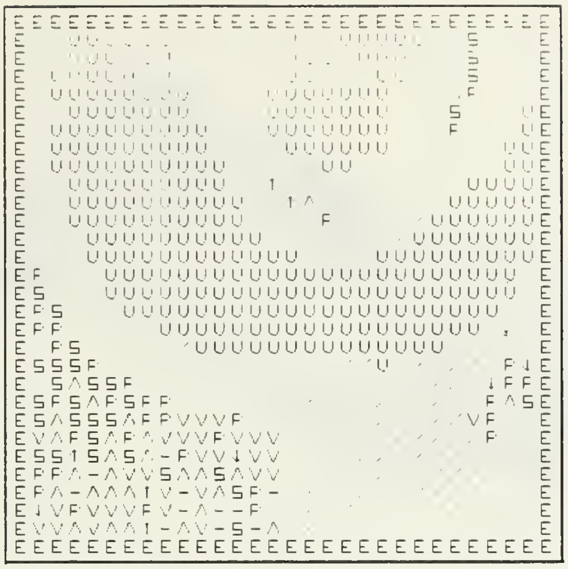

Figure 9. m. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

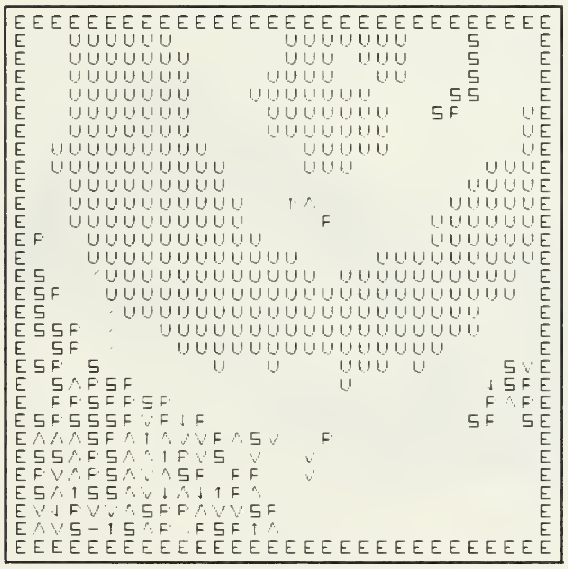

Figure 9. n. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

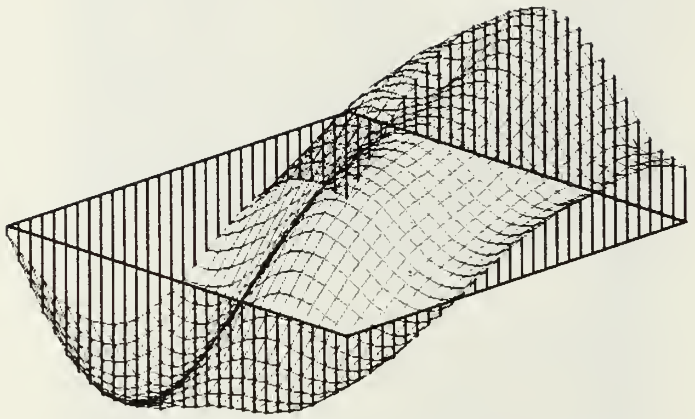

Figure 10. a. Run 4
Clean Terrain

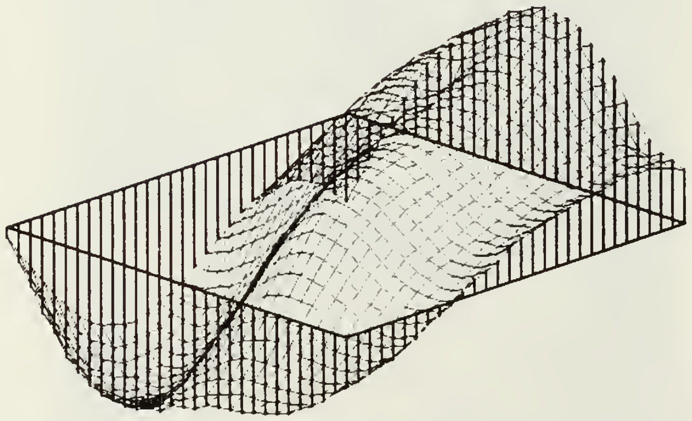

Figure 10. b. Run 4
Noisy Terrain

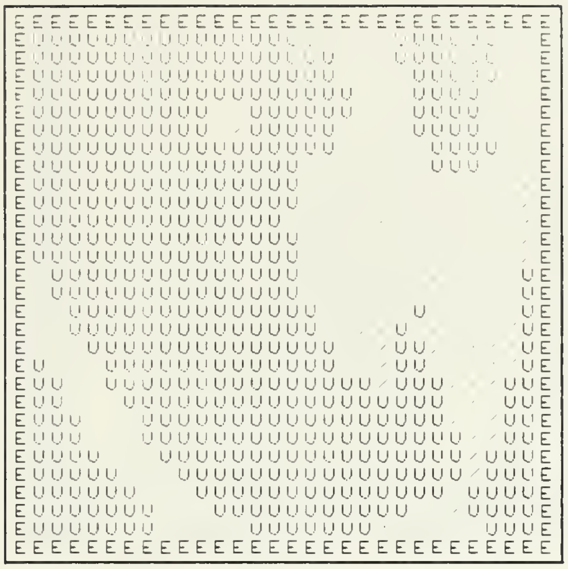

Figure 10. c. Clean Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

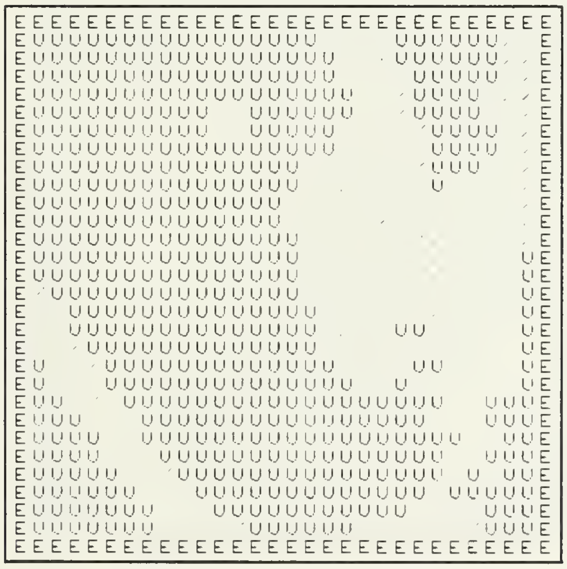

Figure 10. d. Noisy Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

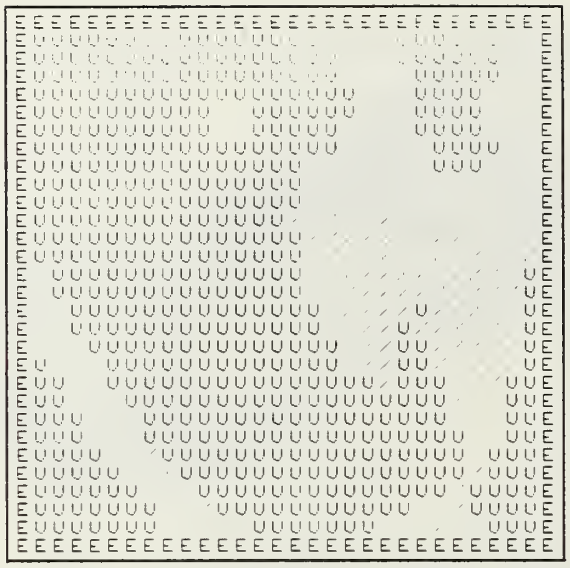

Figure 10. e. Clean Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

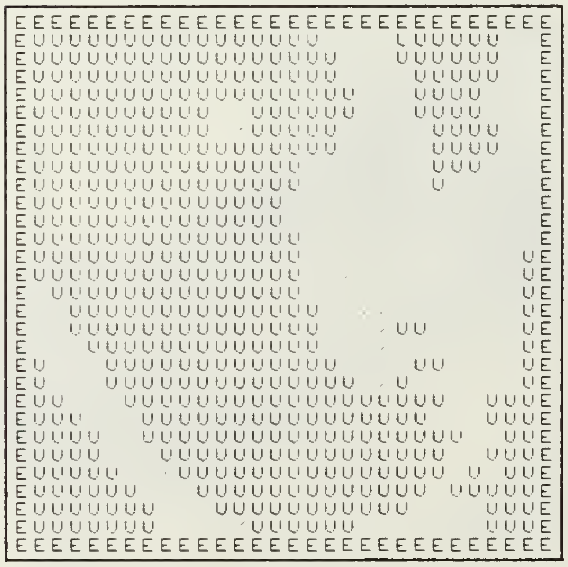

Figure 10. f. Noisy Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

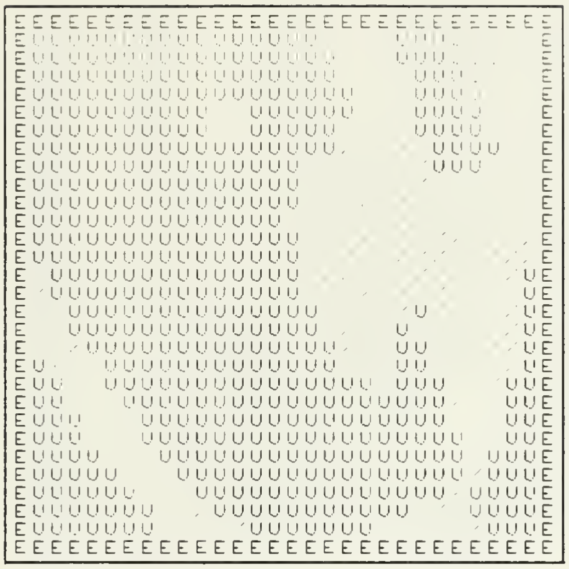

Figure 10. g. Clean Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

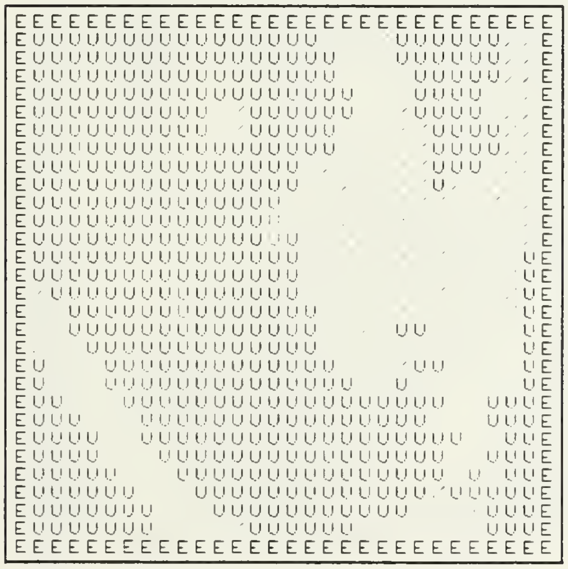

Figure 10. h. Noisy Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

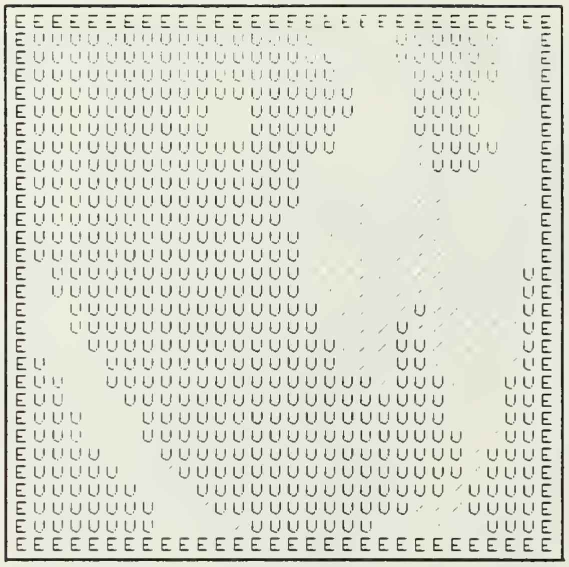

Figure 10. i. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

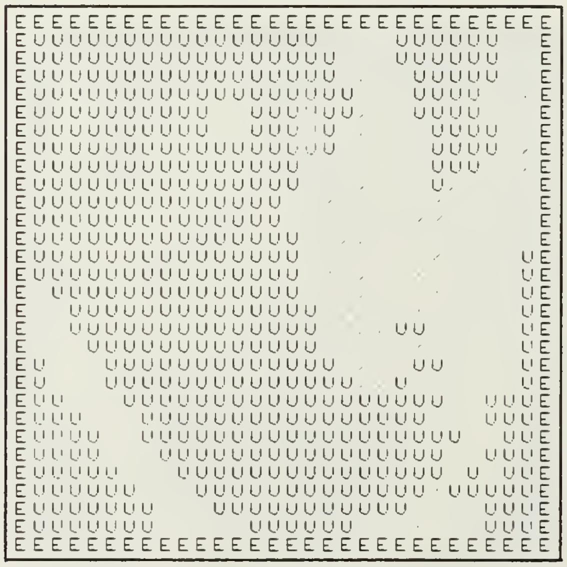

Figure 10. j. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

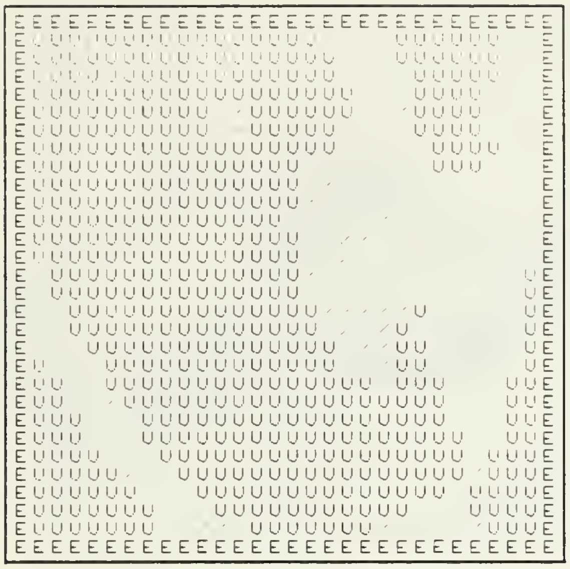

Figure 10. k. Clean Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

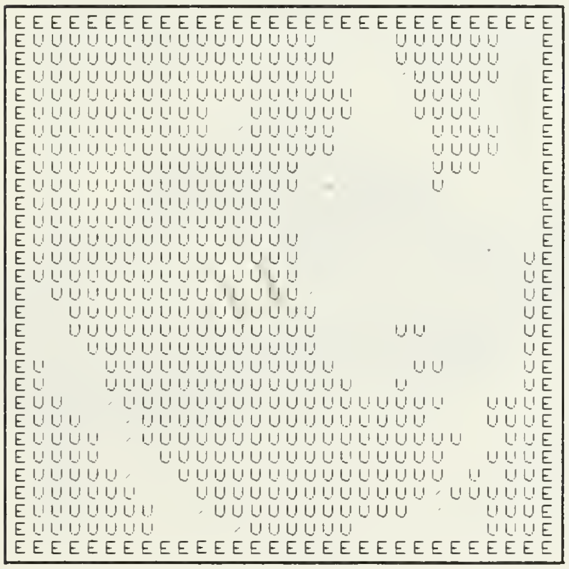

Figure 10. l. Noisy Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

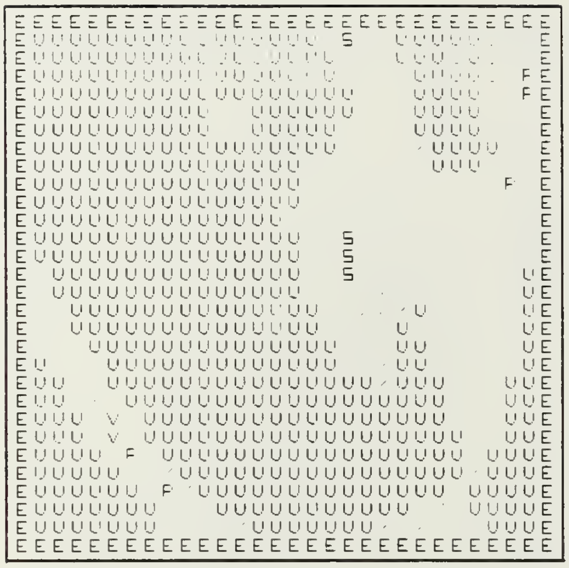

Figure 10. m. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

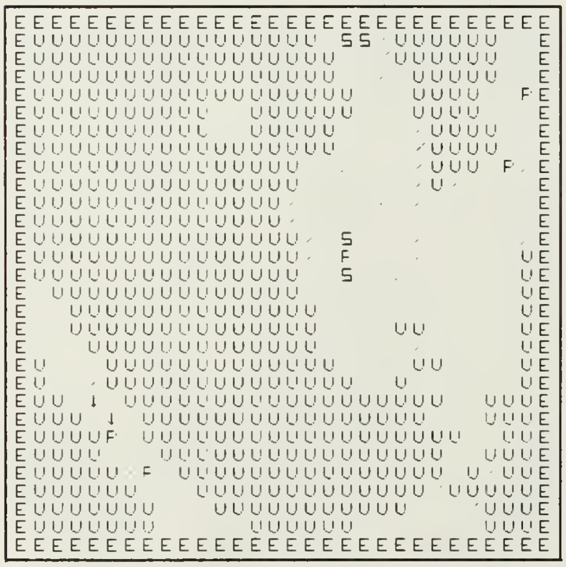

Figure 10. n. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

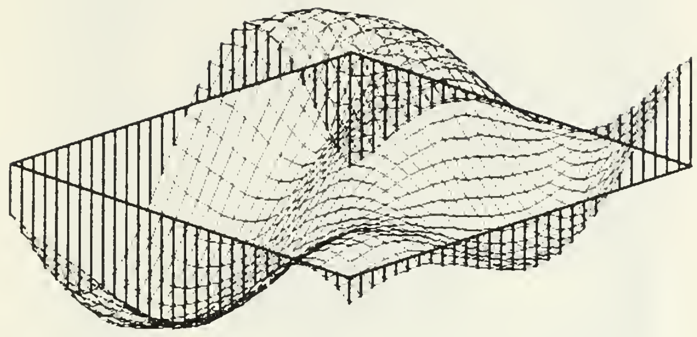

Figure 11. a. Run 5
Clean Terrain

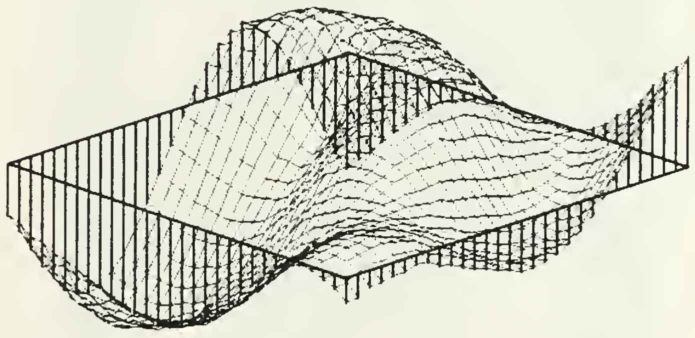

Figure 11. b. Run 5
Noisy Terrain

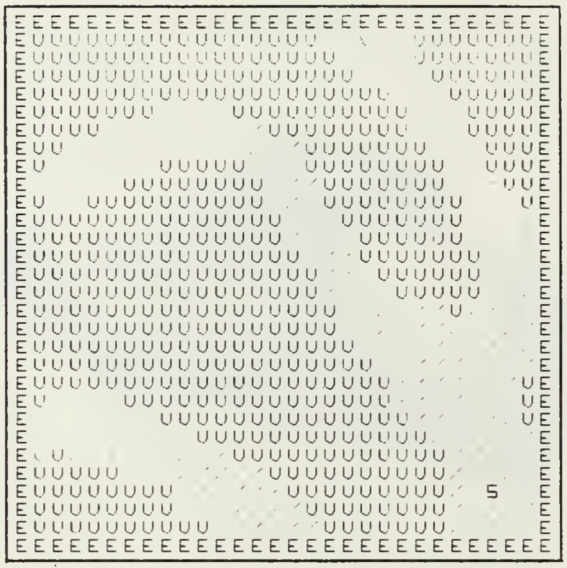

Figure 11. c. Clean Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

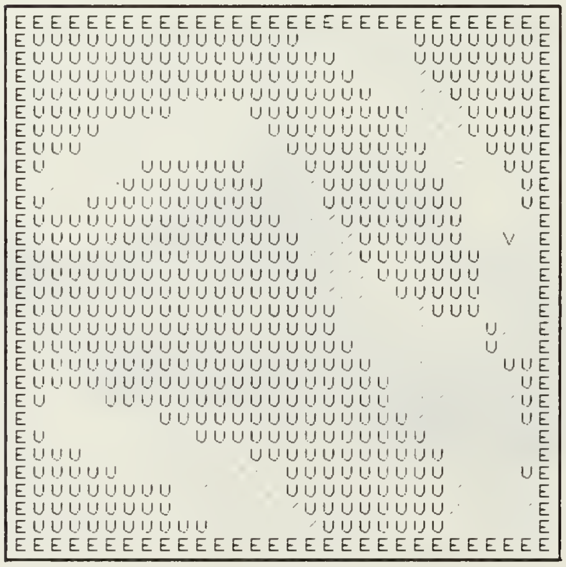

Figure 11. d. Noisy Terrain
Curv_threshold = 0.2 Slope_limit = $2^o$

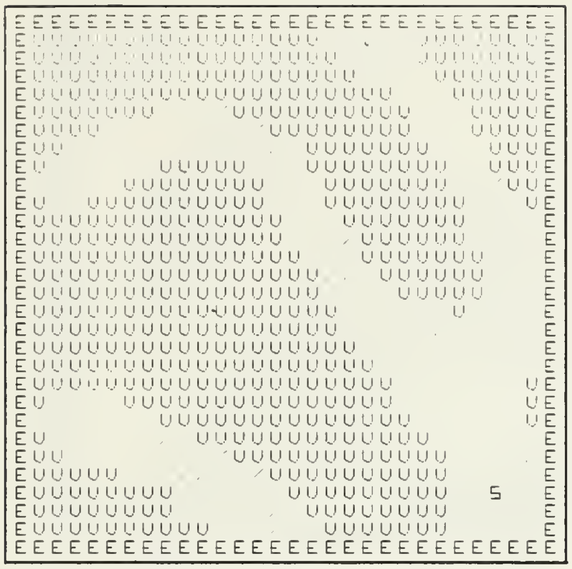

Figure 11. e. Clean Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

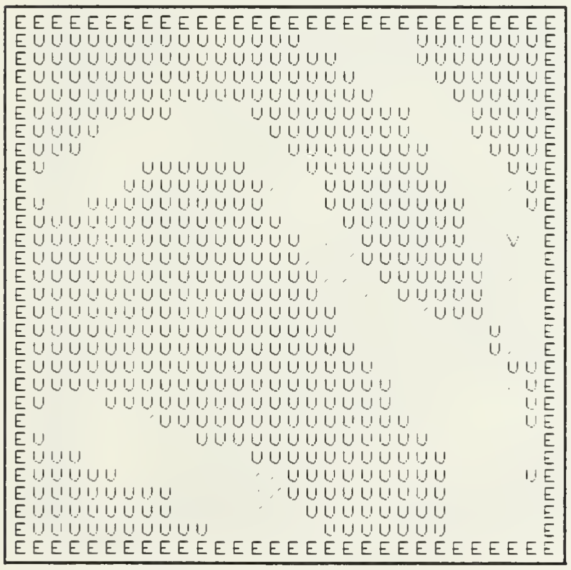

Figure 11.·f. Noisy Terrain
Curv_threshold = 0.15 Slope_limit = $2^o$

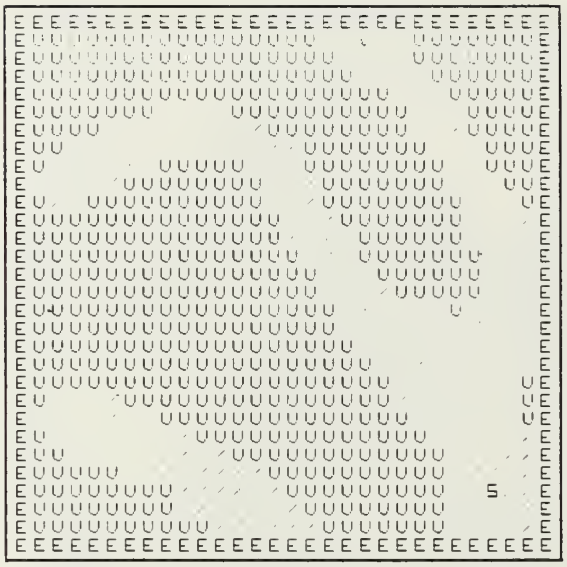

Figure 11. g. Clean Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

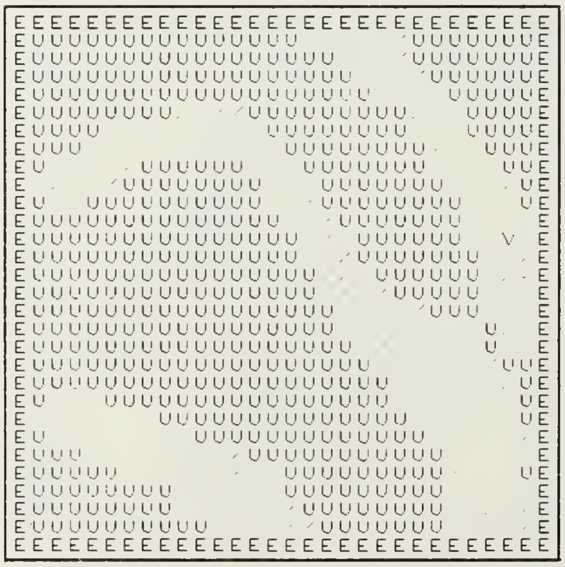

Figure 11. h. Noisy Terrain
Curv_threshold = 0.10 Slope_limit = $2^o$

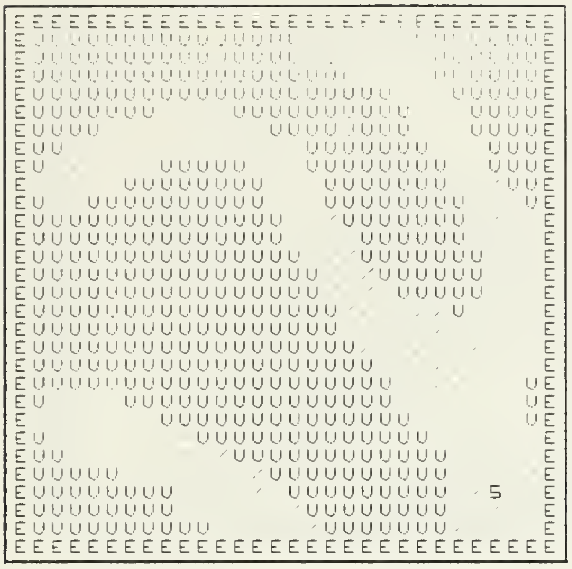

Figure 11. i. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

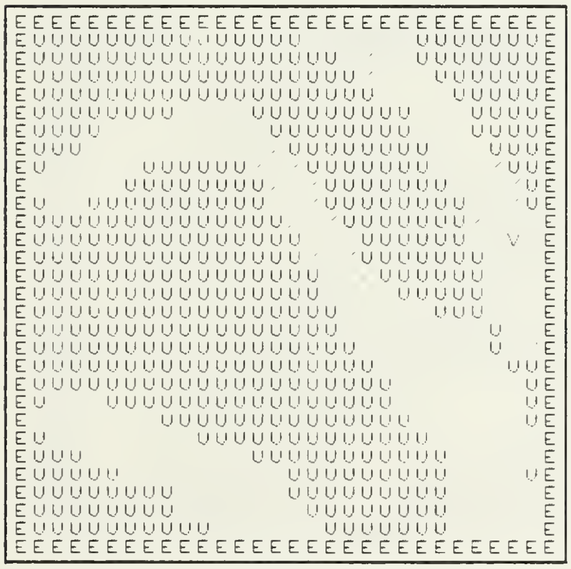

Figure 11. j. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $2^o$

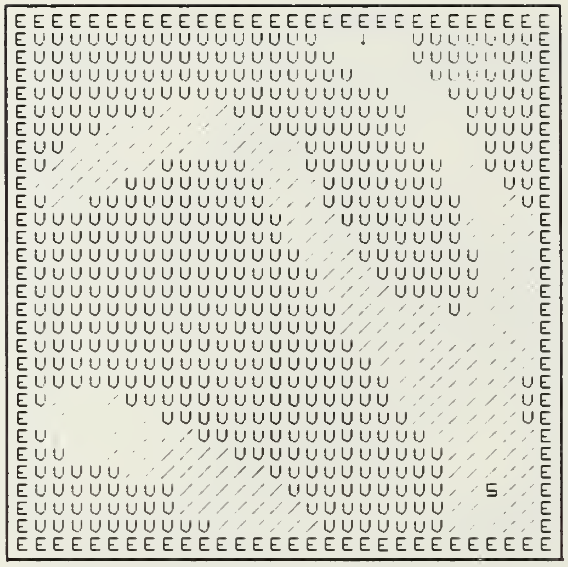

Figure 11. k. Clean Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

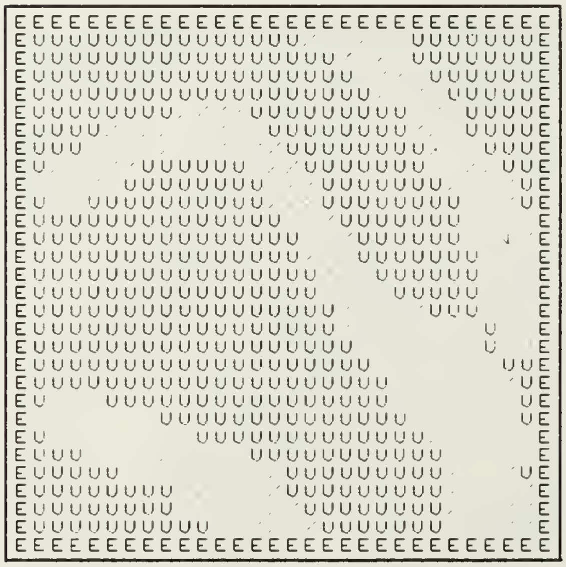

Figure 11. l. Noisy Terrain
Curv_threshold = 0.01 Slope_limit = $2^o$

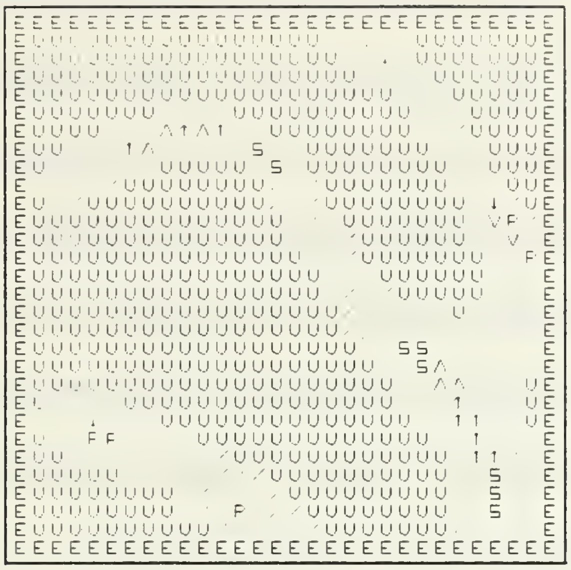

Figure 11. m. Clean Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

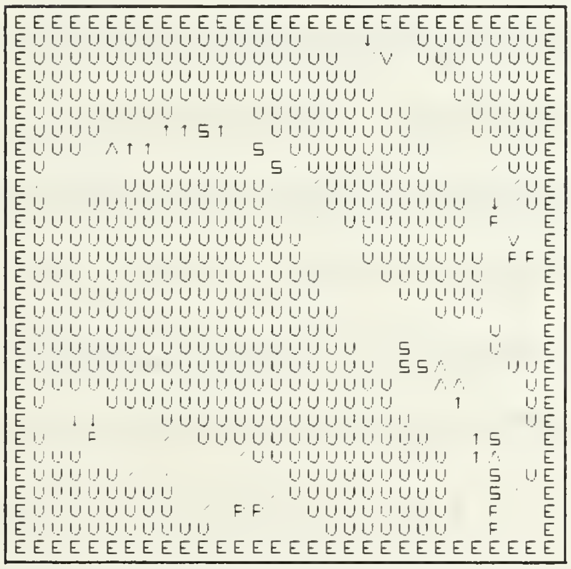

Figure 11. n. Noisy Terrain
Curv_threshold = 0.05 Slope_limit = $10^o$

navigation problem. In Figure 7 all the terrain is either level or safe slope and traversable. Figure 8 is an interesting example in that a second safe route, other than the route up the pass between the two peaks, although not obvious in the terrain picture, is easily seen in the symbolic map. The second route is along the right side of the scene and then up the side of the right peak. The other scenes are not particularly interesting except for the fact that in each case the algorithm produces a symbolic map which can be used to effectively pick out safe routes for the local navigation problem or which, in the case of Figure 11, will allow the autonomous vehicle to decide to not traverse the scene.

## C. THE EFFECT OF CURV_THRESHOLD

The first of the two thresholds to be examined in this sequence of tests is the threshold defined as *curv_threshold* in the program. It is the threshold between a "significant" and an "unsignificant" eigenvalue of the Hessian matrix. This can be though of most easily in terms of the separation between significant and unsignificant curvature on the surface fitted to the window. If the curvature is below the threshold value, it is treated as a zero value when trying to classify terrain as per TABLE 2. For all of the runs, the value of curv_threshold was varied using the following 5 values: 0.2, 0.15, 0.1, 0.05, and 0.01.

Run 1, Figure 7. a-n, is the most illustrative of the effects of the change in curv_threshold. In the other runs the slope_limit over powers curv_threshold. At the largest value for curv_threshold, in run 1, there is very little detail available.

Specifically, in the noiseless terrain, all the level terrain shows up as being flat or planar; i.e. without any curvature. It is not until the curv_threshold decreases to 0.1 that any detail starts to surface. The small peak at the upper left center starts to show as a ridge and the low area at the upper right is classified as valley pixels.

When the curv_threshold is decreased again to 0.05, a significant amount of detail about the scene is brought out. One of the pixels on the small hilltop is classified as a peak terrain type. A hilltop is expected to have peak types surrounded by safe or unsafe slopes. The flat areas are surrounded by valley terrain cells for the most part. A few of the voxels were classified as ridges. An area which starts to curve up in one direction while remaining relatively flat in the other direction will have a positive eigenvalue and so, using TABLE 2, it should be expected that low spots would have valley types surrounding them.

When the curv_threshold is reduced to 0.01 many of the valley and ridge terrain types change to passes and saddles. The difference between ridges and valleys and passes and saddles is that the smaller eigenvalue in magnitude is zero for the ridges and valleys while the eigenvalue is significant for the pass and saddle. Both on the dividing line around the low areas and on the peak, the voxel's which have a pass and saddle classification should probably be considered in error.

Figure 9. a-n is the only other scene which has a significantly sized level area in which the effect of curv_threshold can be examined. This area is sparse and without detail. As the curv_threshold is decreased the voxels take on a random set of classifications. In an area with little detail. a random set of classifications could be expected.

In all the other scenes. the artificial terrain has a significant slope to it, effectively burying the effect of curv_threshold. From the two cases examined, though, is seems that the curv_threshold returns the best information at a value between 0.1 and 0.05. It may be that upon a more detailed examination an optimum value for the curv_threshold may be determinable. It seems more likely that the curv_threshold will have to be varied up and down between about 0.15 and 0.05, depending upon the overall flatness of the scene, to generate good knowledge about the terrain.

## D. THE EFFECT OF SLOPE_LIMIT

To investigate the effect of altering the threshold. slope_limit, the curv_threshold was set to a value of 0.05 which seemed to be a reasonable level from the previous section and the slope_limit was changed from $2^o$ to $10^o$. The effect of this was far more pronounced than the effect encountered in altering curv_threshold.

In Figure 8, not much detail of the scene was evident until the slope_limit was increased. When it was increased. the pass between the two peaks becomes easier to identify. The voxels between the two hills are classified as valleys and depressions for the most part. A valley or depression with unsafe slope on either side of it would be a good way of describing a pass. The rightmost hill is broken out as a ridge voxel. If the curv_threshold were reduced the voxel might show up as a peak cell. It is also interesting to note that the area on the lower left side. where the climbing slope breaks back down. is delineated with a series of pass and saddle voxels. If the curv_threshold were increased in this area, most of the climbing slopes would probably show up as a ridge line. Given this information. an autonomous vehicle would probably identify the area passing between the two unsafe areas as some sort of a pass and, therefore. a usable path.

In Figure 9. the peak is finally broken out when the slope_limit is increased to $10^o$ as is the depression to the right side of the scene. The ridge line in the upper right side is broken out with a voxel being classified as a ridge type along with a series of saddle and pass types. If the curv_limit were increased then these cells would probably degrade to a sequence of ridge type cells.

Though more detail is brought out in Figure 10 with the increase in the slope_limit. the increase is not large enough to bring out significant detail. The only small increase in the detail is along the valley in the lower left corner in which a few of the voxels become valley and pass type cells. An autonomous vehicle transiting this area would probably be able to analyze the scene enough to navigate it based on the secondary characteristics of the safe slope cells. which on this type of symbolic map is not displayed.

In Figure 11, the change in slope_limit brings out considerable detail. The depression in the lower left corner shows up as a depression voxel with a few pass type cells in the neighborhood. The pass in the upper right corner also shows up as a line of valley, pass and depression voxels. The ridge line running from the lower right corner towards the upper left corner also shows up well. Along this line are a number of peak cells and saddle cells with a few ridge cells. With a good set of rules, an autonomous vehicle could probably aggregate that sequence of voxels into the ridge line that it is.

An increase in the slope_limit will not always bring out an effective increase in detail, though. Figure 7 is a good example of how, for a human operator at least, the resultant increase in displayed information with an increase in slope_limit can actually increase the confusion. At a limit of $2^o$ a nice amount of detail is evident. But at a limit of $10^o$ there is too much detail for a human operator to absorb efficiently, though it should be remembered that a computer can effectively utilize much larger amounts of data than can a human. It is interesting to note, however, that whereas the peak in the upper left center area had only displayed a single peak voxel at a $2^o$ slope_limit, there are a number of peak cells in the area when the slope_limit is increased to $10^o$. This would be a very strong indication of an actual peak instead of a false peak due to noise.

## E. THE EFFECT OF NOISE

Each scene is analyzed under both noiseless and noisy conditions. The results of both analyses are presented on the same page for easy comparison. As a general observation, there are two different levels at which the noise effects the analysis. The first level is for the primary classification and the second is for the secondary classification.

### 1. The Effect On Primary Classification

At the level of generating a primary classification, the effect of noise is strongest, as would be expected, at the boundaries between level, safe-slope and unsafe-slope areas. However, at a slope_limit of $2^o$ the level areas are very sensitive to noise as shown in Figures 7 d. and 9 d. where the boundary as well as the interior of the area is effected by the noise. At a slope_limit of $10^o$, though, the effect of noise is mostly on the edges of the area as shown in Figure 9 n. In all the other runs, at both the $2^o$ and $10^o$ slope_limit, the noise has little effect on the safe-slope and unsafe-slope areas except, again, at the boundary. In general, the boundaries are usually displaced only by a single voxel. For the purposes of this algorithm, this level of noise is acceptable since the separation between safe-slope and unsafe-slope can be chosen conservatively and rules can be entered into the navigation package allowing an autonomous vehicle to traverse no closer than 2 (or whatever number is desired) voxels from a particular boundary.

### 2. The Effect On Secondary Classification

The noise has the largest effect on the secondary classification of a voxel. While, in this algorithm, the secondary classification of a voxel is only displayed when the voxel has a primary classification of "level", the effect of noise can still be seen clearly in Run 1 and in the final examples of Runs 2 and 5. In particular, the noise alters the eigenvalues enough so that the algorithm falsely classifies a large number of the voxels. In Run 1. the difference between the noisy and noiseless symbolic maps is significant with the noisy map having a large number of random terrain types while the noiseless map has few. if any, voxels identified as other than "flat" (or planar). In Runs 2 and 5 the noise changes the classification of a number of voxels but the effect is not as significant as when the amplitude of the terrain is relatively flat, as is the case for Run 1. Considering that a pass is a subset of a valley and a saddle is a subset of a ridge, when looking at the final examples of Runs 2 and 5, it appears that the general classification of a voxel is changed little by the noise. A number of valleys turn into passes and a number of ridges turn into saddles, but only in a few cases do valleys turn into their direct opposites, ridges, and peaks into their opposites, pits. It appears that by averaging the terrain types in the local area, the effect of the noise can be partially countered.

## F. CONCLUSIONS

In this chapter, the results obtained from applying the algorithm to a large area, instead of just a single 3x3 window, were examined. The performance of the algorithm was validated in a previous chapter against point sized terrain features, but few major terrain features are point sized.

From the results examined in this chapter, it appears that the first derivative information is of sufficient quality for an autonomous vehicle to navigate. Level and safe slope routes through the scenes were clearly displayed and in one case a non-obvious route was found by the algorithm. The second derivative information added significant detail to the symbolic map. Individual peaks and pits, valleys and ridges, and saddles and passes were highlighted. This information would allow an autonomous vehicle to not only navigate based on safe and unsafe areas but also based on major terrain features such as passes through hills, peak and pits, etc.

The performance of the simulation is very dependent on three variables. The variable first is the slope_limit which is the cut off between level and safe_slope classified terrain. (Another threshold exists between unsafe_slope and safe_slope which was not examined.) The effect of this threshold was examined by varying it between $2^o$ and $10^o$. At the lower threshold, a good amount of detail in the low amplitude run, Run 1, was displayed but little detail in the other runs was displayed. At the upper threshold, the detail overpowered the first derivative information in Run 1 but was good for the other runs. The second variable is curv_threshold which separates the significant and nonsignificant eigenvalues. Generally, the lower the threshold the better the information displayed. This was true until the threshold was lowered down to 0.01 at which point the difference between a valley and a pass and a ridge and a saddle became indistinct. Also, when the terrain was essentially flat, the lower threshold seemed to force another classification on what were really flat or planar voxels. Finally, the effect of noise on the algorithm was examined. Noise appeared to have a smaller effect on the terrain scenes with larger vertical deviations while the scene with little vertical development. Run 1. was strongly effected by the small amount of noise introduced into the signal. In other words. the large amplitude terrain signals overpowered the noise while the low power terrain signals were over powered by the noise.

As can be seen from the results presented in this chapter, the simulation is very dependent on the two parameters, slope_limit and curv_threshold. Before this algorithm can be effectively utilized in an autonomous vehicle, appropriate values for these thresholds will have to be determined. In trying to select values for the thresholds, it may be discovered that a form of adaptive thresholding based on the power in the terrain signal will actually be most effective. How to handle the noise is a more significant problem, though. requiring more study. In general terms, the noise behaves as noise does in an electrical system, overpowering low power signals and being overpowered by large amplitude signals. If normal low pass filtering concepts are employed to help control the noise (such as averaging the voxel types in a local area), then individual hazards (such as a rock in a vehicle's path which would show up as a single peak voxel) may be filtered out. Clearly, noise is a difficult problem requiring careful analysis and a much more work than the scope of this thesis permitted.

# VI. SUMMARY AND CONCLUSIONS

Any autonomous vehicle, whether it is a land vehicle walking the hills of Colorado or a space probe exploring the planet Mars, requires the ability to sense it's environment. If the vehicle operates at a distance from it's target, such as the Voyager probes sensing Saturn at thousands of kilometers or a Cruise Missile flying at a predetermined altitude, then the sensors do not require extremely high resolution. Sensors operating in the IR region, Radar, and Ultrasound may be good enough in those cases. However, if the vehicle operates in very close proximity to a hostile environment, such as by walking on it, then it is advantageous to sense the environment with high resolution. This implies utilizing sensors operating in the optical regime.

To date, most research in optical wavelength sensing has concentrated on two-dimensional processing to generate intrinsic images or object edges. Little research had been conducted on three-dimensional optical sensing for object classification, which is more useful when considering the autonomous vehicle navigation problem, until the work done by Olivier at Ohio State University. The purpose of this thesis, has been to do a detailed mathematical development of the formulas utilized by Olivier and then to write a simulation, based on those formulas, which attempts to analyze large random geographic terrain scenes as a proof of the concept of terrain classification for autonomous vehicle navigation.

## A. RESEARCH CONTRIBUTIONS

In this thesis, a number of advances have been made over the work originally publisher by Olivier in his masters thesis. The primary advance is the mathematical derivation of two new terrain types. These are the *pass* and the *saddle*. The pass is a subtype of the *valley* in which, instead of the smaller eigenvalue of the Hessian matrix being zero, it is negative while the larger eigenvalue is positive. The saddle is a subset of the *ridge* in which, instead of the smaller eigenvalue being zero, the smaller eigenvalue is positive while the larger eigenvalue is negative. These two new terrain classifications resulted from a mathematical study of the theory of quadratic surfaces. When combined with the five other types previously defined in Olivier's work, a complete set of all the possible terrain classifications, which can be made based solely on the values of the eigenvalues of the Hessian matrix, is formed.

Secondly, a complete mathematical derivation of the formulas on which the classification of terrain voxels is based was presented. The formulas are dependent on the sampling points being regularly spaced about the voxel of interest at $\pm$ one distance unit. The particular distance unit is not important; it can be meters, centimeters, or kilometers depending on the scale of the scene being analyzed. The effect of the regular spacing of the sampling points is the elimination of a number of terms in the six equations resulting from a least squares error analysis of a quadratic equation describing a three dimensional surface. When the six equations are simplified, through the elimination of those terms. five equations are derived which allow the constants of the quadratic to be found. These constants can then be used to easily calculate the gradient and eigenvalues of the Hessian matrix for the least square surface. which in turn. are the values which are the basis of the classification scheme.

Finally, the algorithm. including the two new terrain classification types. was implemented. in LISP (one of the two main languages used today in artificial intelligence research). and ran against terrain scenes with relatively realistic major terrain features. The output of the simulation was designed to graphically display the capability of the algorithm to produce symbolic results which could be used by an autonomous vehicle to find safe paths through the scene as well as identify major terrain features for the purposes of navigation planning. From the results of the five simulation runs. it is clear that the algorithm is capable of analyzing large scale scenes. It is also clear that the performance of the algorithm is very dependent on the proper selection of appropriate thresholds for the roughness of the terrain being analyzed. If the slope_limit is too low or the curv_limit is too high. the amount of knowledge generated by the simulation is degraded. If the two thresholds are set to be too sensitive, the information generated is of questionable value. The effect of noise on the performance of the simulation was also examined and found to be significant depending on the "power" in the terrain signal.

Another major result of this work, though not related specifically to the algorithm, is the generation of a number of routines to handle graphics in LISP on the ISI Optimum V Workstation. Franz LISP. the variant of LISP implemented on the workstations. has no inherent graphics capabilities. This is a major limitation considering the outstanding graphics capabilities of the workstations. By "importing" these graphics routines into LISP (through the LISP cfasl command), new LISP functions can be generated which will allow Franz LISP to utilize most of the graphics capabilities of the workstations.

## B. RESEARCH EXTENSIONS

This simulation was developed primarily as a proof of concept of the usefulness of the idea of terrain classification based on range images. As such it has been very successful. There are a number of problems, though. with the simulation which require further study before the algorithm can be employed in an autonomous vehicle. First, the algorithm is relatively slow. This simulation is installed as an interperted set of routines and has not been compiled successfully. As such, it takes about 20 minutes to complete one scene analysis. This is prohibitively long for an autonomous vehicle. Therefore, the classification portion of the simulation needs to be compiled on a LISP machine or possibly installed in machine language before it can be used in real time. It may also be possible to significantly improve the speed of the simulation by writing the mathematically intensive portions of the algorithm in Fortran or Pascal, compiling them separately, and then accessing them, in the LISP environment, through the cfasl command.

Another important area for further study is attempting to utilize the voxel classifications in aggregating individual voxels into major terrain features. It is easy enough for a human to identify ridge lines as a series of ridge voxels and hills as peak voxels surrounded by safe-slope voxels, but this a capability the algorithm does not now possess. By examining the symbolic results in Chapter 5, it is obvious that aggregation will involve more that just combining like terrain types. Depending on the thresholding, for example, valley voxels can be displayed as either valleys, passes, or even flat or planar voxels. The same is true of all the other terrain types. Also, there are a number of other factors that need to be considered when attempting to aggregate and identify major terrain features. An example is the angle between the principal eigenvector and the gradient of a voxel. As was briefly discussed before, a valley line oriented up a slope is probably just a ravine, whereas, if it is oriented perpendicular to the slope it could be a path. The possible variation between slope, the eigenvalues, the eigenvectors, as well as the classification of nearest neighbors leads to a large number of possibilities. An aggregation scheme based on this classification algorithm is a good example of a system which would respond well to the techniques of artificial intelligence wherein a number of general rules, based on a some of the factors, could be devised to classify each voxel.

Another good way of trying to aggregate, instead of attempting to write long lists of rules which cover each individual case, would be to let a robot learn what is important when aggregating voxels by exploring a wide variety of terrain scenes and then have the robot write its own set of rules. An actual robot in a natural environment would not be specifically required. Instead a computer simulation of the dynamics of a robot or autonomous vehicle could be written which would then drive or walk across a number of artificially created terrain scenes. By calculating the energy output of the dynamic system and also recording the events which lead to catastrophic situations and then comparing them to terrain classifications derived through this model, the robot could "learn" to effectively utilize the information generated in this simulation. A well written simulation in LISP could potentially even write its own set of rules for both aggregation and path planning which the human could access to see what the robot "thinks" is important.

The effect of noise is well worth further research. It was only touched upon in this thesis but it is obvious that a form of filtering must be developed if the algorithm is to be installed in truly autonomous vehicles. A form of low pass filtering, in which neighbors are averaged, may be useful but it appears that such filtering could destroy knowledge about small hazards. Another possibility is a form of Kalman filtering or time averaging where random noise will show up as random classifications in subsequent sweeps of the sensor, unlike a true hazard which will remain relatively constant over time.

There are numerous of other possible research extensions which can evolve from this simulation. Adaptive thresholding for the slope_limit and curv_threshold and extending this algorithm to third order or higher surfaces are but a few. In actuality this algorithm has been developed more as a tool which will hopefully be expanded upon than as an actual algorithm to be implemented in a working system. As such it exhibits a good potential to be the basis for giving an autonomous vehicle a reasonable vision capability for terrain recognition.

# APPENDIX A – MATRIX DERIVATION OF COEFFICIENTS

This is a second derivation to find the formulas for the coefficients of a quadratic least squares fit to a three dimensional surface. This derivation is based on matrix calculus. First it is necessary to assume, as in the previous derivation, that the data will be in a regularly spaced 3x3 window as in TABLE 1. The ordering of the elevation data is as in TABLE 9.

TABLE 9

ORDERING OF ELEVATION DATA

| | | |
|---|---|---|
| $z_1$ | $z_2$ | $z_3$ |
| $z_4$ | $z_5$ | $z_6$ |
| $z_7$ | $z_8$ | $z_9$ |

The estimating equation for the elevation data is:

$$\hat{z}_i = k_1 + k_2 x_i + k_3 y_i + k_4 x_i^2 + k_5 x_i y_i + k_6 y_i^{2.} \tag{A.1}$$

The error between the real elevation $z$ and $\hat{z}$, $\epsilon$, is:

$$\epsilon_i = (z - z_i) \tag{A.2}$$

and the sum of the squared errors,$\Phi$, is:

$$\Phi = \sum \epsilon_i^2 = \epsilon^T \epsilon. \tag{A.3}$$

In vector matrix form. Equation A.1 is:

$$\hat{z}_{9x1} = A_{9x6}\, k_{6x1} \tag{A.4}$$

where the subscripts are the size of the matrices.

$$k = (\, k_1\ k_2\ k_3\ k_4\ k_5\ k_6\, )^T \tag{A.5}$$

and

$$A_i = (\, 1\ x_i\ y_i\ x_i^2\ x_i y_i\ y_i^2\, ). \tag{A.6}$$

The squared errors are then, in matrix form:

$$\Phi = (\hat{z} - z)^T\, (\hat{z} - z) \tag{A.7}$$

$$= (z - Ak)^T\, (z - Ak) \tag{A.8}$$

$$= (z^T - k^T A^T)\, (z - Ak) \tag{A.9}$$

$$= z^T z - k^T A^T z - z^T A k + k^T A^T A k. \tag{A.10}$$

Letting the product:

$$A^T z = w \tag{A.11}$$

the two middle terms can be replaced by

$$k^T w \tag{A.12}$$

and

$$w^T k \tag{A.13}$$

respectively. Equations A.12 and A.13 are equal scalar dot product expressions and so Equation A.10 may be written as:

$$= z^T z - 2k^T A^T z + k^T A^T A k. \tag{A.14}$$

The gradient of the error is:

$$\nabla\Phi = \left[ \frac{\partial \Phi}{\partial k_i} \right] = -2A^T z + 2A^T A k = 0 \tag{A.14}$$

or

$$A^T A k = A^T z. \tag{A.15}$$

Thus:

$$k = \left[ A^T A \right]^{-1} A^T z \tag{A.16}$$

or

$$= Bz \tag{A.17}$$

where

$$B = \left[ A^T A \right]^{-1} A^T \tag{A.18}$$

and

$$A = \begin{vmatrix} 1 & -1 & 1 & 1 & -1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 0 & 0 & 1 \\ 1 & 1 & -1 & 1 & -1 & 1 \end{vmatrix} \tag{A.19}$$

To find the coefficients of the quadratic it is only necessary then to calculate the B matrix and then multiply it by the z matrix. To calculate the B matrix, the transpose of the A matrix must be produced so that it may be multiplied by the A matrix. The transpose of the A matrix is:

$$A^T = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix} \tag{A.20}$$

The product of A transpose and A is:

$$A^T A = \begin{bmatrix} 9 & 0 & 0 & 6 & 0 & 6 \\ 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 \\ 6 & 0 & 0 & 6 & 0 & 4 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 6 & 0 & 0 & 4 & 0 & 6 \end{bmatrix} \tag{A.21}$$

This product is inverted and then multiplied by A transpose, Equation A.20. The inverted matrix is:

$$(A^T A)^{-1} = \begin{bmatrix} 5/9 & 0 & 0 & -1/3 & 0 & -1/3 \\ 0 & 1/6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/6 & 0 & 0 & 0 \\ -1/3 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 0 \\ -1/3 & 0 & 0 & 0 & 0 & 1/2 \end{bmatrix} \tag{A.22}$$

The product of the inverted term and A transpose is the B matrix, which is:

$$B = \begin{bmatrix} -1/9 & 2/9 & -1/9 & 2/9 & 5/9 & 2/9 & -1/9 & 2/9 & -1/9 \\ -1/6 & 0 & 1/6 & -1/6 & 0 & 1/6 & -1/6 & 0 & 1/6 \\ 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & -1/6 & -1/6 & -1/6 \\ 1/6 & -1/3 & 1/6 & 1/6 & -1/3 & 1/6 & 1/6 & -1/3 & 1/6 \\ -1/4 & 0 & 1/4 & 0 & 0 & 0 & 1/4 & 0 & -1/4 \\ 1/6 & 1/6 & 1/6 & -1/3 & -1/3 & -1/3 & 1/6 & 1/6 & 1/6 \end{bmatrix} \tag{A.23}$$

To calculate the constants of the quadratic least squares surface, the above matrix, B, is multiplied by the 1x9 column vector of elevation data, z. The resulting 1x6 column vector are the 6 constants, $k_1$ through $k_6$.

# APPENDIX B – GRAPHICS FUNCTIONS

```
/*****************************************************
/*                                          */
/*              TOFILE                      */
/*                                          */
/*  This is a C routine compiled separately     */
/*  which is designed to read a bitmap from     */
/*  the ISI Graphics machines. It is intended    */
/*  to be compiled in line in LISP environment   */
/*  to provide graphics calls. The function     */
/*  is called "tofile" from LISP. It is called   */
/*  as (tofile bmd filename) and it will        */
/*  return a t when complete. The parameters     */
/*  passed to this "C" functions are:           */
/*                                          */
/*  bmd  =   bitmap descriptor number          */
/*                                          */
/*  filename  =   the file name that the        */
/*              bitmap is to be save          */
/*              under                       */
/*                                          */
/*  The way this function works is by           */
/*  calling a C function which reads the        */
/*  bits in the bitmap, 4 at a time. If the     */
/*  first bit is a 1, then 8 is added to        */
/*  the counter bitval. If the next bit is      */
/*  set then a 4 is added to bitval. Etc...     */
/*  until the value of all 4 bits is added      */
/*  to bitval. It then goes down a list,        */
/*  comparing the value of bitval until         */
/*  it matches it. The letter or decimal        */
/*  value that that equates to in hexa-         */
/*  decimal is then printed to the file         */
/*  under filename. The junk it prints to       */
/*  filename at the start and finish of the     */
/*  program are control codes to the laser      */
/*  printer which allow the bitmap code         */
/*  to be printed. It should be noted           */
/*  that the area of the the bit map that       */
/*  is read is 576 by 448 bits. This is         */
/*  one entire window in this simulation.       */
/*  If it is desired to use this program        */
/*  to read a bit map of a different size,      */
/*  the area read must be in blocks of 64       */
/*  bits long. Also, the first numbers printed  */
/*  in the file after the control codes must    */
/*  a 4 digit decimal number which is the       */
/*  number of bits long each row to be          */
/*  printed is. After that. all the other        */
/*  numbers are the hex code of the bitmap.       *
/*                                          *
/**********************************************
/*                                          *
/**********************************************/
/*                                          */
/*       Dennis Poulos                           */
/*       x 3403                              */
/*       26 May 1986                            */
/*                                          */
/**********************************************/
#include <stdio.h>
#include <vt.h>
#include <bitmap.h>
#include <math.h>

FILE *fopen(),*fp;

tofile (bmp,filename)
    char *filename[];
    int *bmp;
{
    int temp;
    struct BMD *bmd;
    short ii,j,k,bitv;
    int bitval;
    short boxw,boxh;
    temp = *bmp;
    bmd = temp;
    fp=fopen(filename,"w");
    fprintf(fp,"^PY^-0F^G0);
    fprintf(fp,"^IJ01500^IT01000);
    fprintf(fp,"^,^G0SM11000IGV^PW050IGE");
    fprintf(fp,"^IP02020);
    boxw=576;
    boxh=448;
    fprintf(fp,"^P");
    fprintf(fp,"0");
    fprintf(fp,"%d",boxw);
    for(ii=0;ii<boxh;ii++)
        {
        for(j=0;j<boxw;j+=4)
            {bitval = bitv = 0;
            if((bitv=BM_ReadBit(bmd,j,ii))==0)
                bitval += 8;
            if((bitv=BM_ReadBit(bmd,(j+1),ii))==0)
                bitval += 4;
            if((bitv=BM_ReadBit(bmd,(j+2),ii))==0)
                bitval += 2;
            if((bitv=BM_ReadBit(bmd,(j+3),ii))==0)
                bitval += 1;
            if(bitval == 15)
                fprintf(fp."F"):
            else if(bitval == 14)
                fprintf(fp."E"):
            else if(bitval == 13)
                fprintf(fp,"D"):
            else if(bitval == 12)
                fprintf(fp,"C"):
            else if(bitval == 11)
                fprintf(fp,"B");
            else if(bitval == 10)
                fprintf(fp,"A");
            else
                fprintf(fp,"%d",bitval);
            }
        fprintf(fp,"0);
        }
    fprintf(fp," ^.0G0);
    fprintf(fp," ^O ^G0- ^PN ^-0);
    fclose(fp);
    return;
}
/***************************************************/
/*                                              */
/*   compile as                                   */
/*   cc -c tofile.c -ltools -lbm -lvt               */
/*                                              */
/***************************************************/
```

```
c     ************************************************
c     *                                              *
c     *              OPEN-WINDOW                     *
c     *                                              *
c     *   This is a fortran subroutine compiled      *
c     *   separately to open a window on the ISI     *
c     *   Graphics Machines. It is intended to be    *
c     *   compiled inline in a LISP environment to   *
c     *   provide LISP graphics calls. The function  *
c     *   is called "opw". From LISP it is called    *
c     *   as (opw wx wy ww wh "title") and it will   *
c     *   return the fdnum (file descriptor number)  *
c     *   when complete. The parameters passed to    *
c     *   the fortran function from LISP are:        *
c     *                                              *
c     *   wx   =   x coordinate of upper left corner *
c     *                                              *
c     *   wy   =   y coordinate of upper left corner *
c     *                                              *
c     *   ww   =   window width                      *
c     *                                              *
c     *   wh   =   window height                     *
c     *                                              *
c     *   title  =  window title (max 31 characters) *
c     *                                              *
c     *   all coordinates are in numbers of pixels   *
c     *                                              *
c     ************************************************
c
c     ************************************************
c     *                                              *
c     *       Dennis Poulos                          *
c     *       x 3403                                 *
c     *       15 April 1986 (version 2)              *
c     *                                              *
c     ************************************************
c
      integer function opw(wx,wy,ww,wh,title)
         external OpenWindow
         integer OpenWindow,wx,wy,ww,wh,opw
         character*31 title
      opw =   OpenWindow(wx.wy.ww,wh,title)
      return
      end
c
c     ************************************************
c     *                                              *
c     *   compile as                                 *
c     *   f77 -c openw.f -ltoolsf -lbmf -lvtf        *
c     *                                              *
c     ************************************************
```

```
c     ************************************************
c     *                                              *
c     *          LINE-DISCIPLINE                     *
c     *                                              *
c     *  This is a fortran subroutine compiled       *
c     *  separately to set the line discipline       *
c     *  of the window associated with the file      *
c     *  descriptor fd. The default line discipline  *
c     *  of a window is NTTYDISC. To access the      *
c     *  graphics capabilities of the ISI the        *
c     *  line discipline must be set to TWSDISC.     *
c     *  The function is called "slined". From LISP  *
c     *  it is called as (slined fd) and it will     *
c     *  return a "t" when complete. The parameters  *
c     *  passed to the fortran function from LISP    *
c     *  are:                                        *
c     *                                              *
c     *  fd  =  file descriptor of the desired       *
c     *           window                             *
c     *                                              *
c     ************************************************
c
c     ************************************************
c     *                                              *
c     *       Dennis Poulos                          *
c     *       x 3403                                 *
c     *       15 April 1986                          *
c     *                                              *
c     ************************************************
c
      integer function slined(fd,mode)
         external SetLineDisc
         integer fd,slined,mode,SetLineDisc
      slined = SetLineDisc(fd,mode)
      return
      end
c
c     ************************************************
c     *                                              *
c     *  compile as                                  *
c     *  f77 -c setldisc.f -ltoolsf -lbmf -lvtf      *
c     *                                              *
c     ************************************************
```

```
c     ************************************************
c     *                                              *
c     *          BIT MAP ALLOCATE                    *
c     *                                              *
c     *  This is a fortran subroutine compiled       *
c     *  separately to allocate a portion of memory  *
c     *  for the bit map. A bit map is maintained    *
c     *  in memory and can be displayed in a         *
c     *  window as desired. This routine is intended *
c     *  to be compiled inline in a LISP envir-      *
c     *  onment to provide LISP graphics calls.      *
c     *  The function is called "bitmapa". From      *
c     *  LISP it is called as (bitmapa width height) *
c     *  and it will return the bmd (bit map         *
c     *  descriptor) when complete. The parameters   *
c     *  passed to the fortran function from LISP    *
c     *  are:                                        *
c     *                                              *
c     *  width    =    bit map width                 *
c     *                                              *
c     *  height   =    bit map height                *
c     *                                              *
c     *  all coordinates are in numbers of pixels    *
c     *                                              *
c     ************************************************
c
c     ************************************************
c     *                                              *
c     *      Dennis Poulos                           *
c     *      x 3403                                  *
c     *      18 April 1986                           *
c     *                                              *
c     ************************************************
c
      integer function bitmapa(width,height)
         external bmAllocate
         integer bmAllocate,bitmapa,width,height
         bitmapa=bmAllocate(width,height)
      return
      end
c
c     ************************************************
c     *                                              *
c     *  compile as                                  *
c     *  f77 -c bitmal.f -ltoolsf -lbmf -lvtf        *
c     *                                              *
c     ************************************************
```

```
c    *************************************************
c    *                                               *
c    *         ADDRESSING MODE                       *
c    *                                               *
c    *  This is a fortran subroutine compiled        *
c    *  separately to set the display mode in a      *
c    *  window. The two modes available are          *
c    *  "VT_RELATIVE" AND "VT_ABSOLUTE". Relative    *
c    *  addressing means pixels are specified        *
c    *  relative to the current pointer position     *
c    *  while absolute addressing specifies pixels   *
c    *  relative to the upper left hand corner       *
c    *  of the bitmap. This routine is intended      *
c    *  to be compiled inline in a LISP environment  *
c    *  to provide LISP graphics calls. The function *
c    *  is called as (bmadres bmd mode) and it will  *
c    *  return a "t" when complete. The parameters   *
c    *  passed to the fortran subroutine from        *
c    *  LISP are:                                    *
c    *                                               *
c    *  bmd  =   bitmap descriptor number            *
c    *                                               * .
c    *  mode =   addressing mode                     *
c    *                                               *
c    *************************************************
c
c    *************************************************
c    *                                               *
c    *      Dennis Poulos                            *
c    *      x 3403                                   *
c    *      20 April 1986                            *
c    *                                               *
c    *************************************************
c
     subroutine bmadres(bmd,mode)
        integer bmd,mode
        call bmSetAddress(bmd,mode)
     return
     end
c
c    *************************************************
c    *                                               *
c    *  compile as                                   *
c    *  f77 -c bmadres.f -ltoolsf -lbmf -lvtf        *
c    *                                               *
c    *************************************************
```

```
c     *************************************************
c     *                                               *
c     *           SET BACKGROUND COLOR                   *
c     *                                               *
c     *  This is a fortran subroutine compiled        *
c     *  separately to set the background color       *
c     *  of a bitmap. The background color can         *
c     *  be set to either VT_White or VT_BLACK.         *
c     *  This routine is intended to be compiled       *
c     *  inline in a LISP environment to provide        *
c     *  LISP graphics calls. The function is called    *
c     *  as (setbacc bmd color) and it will            *
c     *  return a "t" when complete. The               *
c     *  parameters passed to the fortran subroutine   *
c     *  from LISP are:                           *
c     *                                               *
c     *  bmd  =    bitmap descriptor number          *
c     *                                               *
c     *  color =   VT_White                      *
c     *              or                          *
c     *           VT_Black                       *
c     *                                               *
c     *************************************************
c
c     *************************************************
c     *                                               *
c     *      Dennis Poulos                         *
c     *      x 3403                           *
c     *      22 April 1986                         *
c     *                                               *
c     *************************************************
c
      subroutine setbacc(bmd,color)
         integer bmd,color
         call bmSetBColor(bmd,color)
      return
      end
c
c     *************************************************
c     *                                               *
c     *  compile as                          *
c     *  f77 -c setbacc.f -ltoolsf -lbmf -lvtf        *
c     *                                               *
c     *************************************************
```

```
c     **************************************************
c     *                                                *
c     *               SET COLOR                       *
c     *                                                *
c     *  This is a fortran subroutine compiled        *
c     *  separately to set the color objects are      *
c     *  drawn in to the bitmap. The color can        *
c     *  be set to any of a number of shade of        *
c     *  grey along with black or white. For this     *
c     *  simulation the colors will usually           *
c     *  be set to either VT_White or VT_BLACK.       *
c     *  This routine is intended to be compiled      *
c     *  inline in a LISP environment to provide      *
c     *  LISP graphics calls. The function is called  *
c     *  as (setcol bmd color) and it will            *
c     *  return a "t" when complete. The              *
c     *  parameters passed to the fortran subroutine  *
c     *  from LISP are:                               *
c     *                                                *
c     *  bmd  =   bitmap descriptor number            *
c     *                                                *
c     *  color =  VT_White                .           *
c     *              or                               *
c     *          VT_Black                             *
c     *                                                *
c     **************************************************
c
c     **************************************************
c     *                                                *
c     *      Dennis Poulos                            *
c     *      x3403                                    *
c     *      22 April 1986                            *
c     *                                                *
c     **************************************************
c
      subroutine setcol(bmd.color)
         integer bmd.color
         call bmSetColor(bmd.color)
      return
      end
c
c     **************************************************
c     *                                                *
c     *  compile as                                   *
c     *  f77 -c setcol.f -ltoolsf -lbmf -lvtf         *
c     *                                                *
c     **************************************************
```

```
c     **************************************************
c     *                                                *
c     *              CLEAR BITMAP                      *
c     *                                                *
c     *  This is a fortran subroutine compiled         *
c     *  separately to clear a bitmap after it         *
c     *  has been drawn on. It will clear the          *
c     *  region of the bitmap pointed to by bmd        *
c     *  from the position x and y for a width         *
c     *  w and height h. It will clear the region      *
c     *  by coloring the entire space in the           *
c     *  color specified. These values are passed      *
c     *  as parameters. This fortran subroutine        *
c     *  is intended to be compiled                    *
c     *  inline in a LISP environment to provide       *
c     *  LISP graphics calls. The function is called   *
c     *  as (bmclear bmd x y w h color). It will       *
c     *  return a "t" when complete. The               *
c     *  parameters passed to the fortran subroutine   *
c     *  from LISP are:                                *
c     *                                                *
c     *  bmd  =   bitmap descriptor number             *
c     *                                                *
c     *  x    =   x coordinate of upper left           *
c     *           corner of region                     *
c     *                                                *
c     *  y    =   y coordinate of upper left           *
c     *           corner of region                     *
c     *                                                *
c     *  w    =   width of area to be cleared          *
c     *                                                *
c     *  h    =   height or area to be cleared         *
c     *                                                *
c     *  color =  color to set the region to           *
c     *           to clear it.                         *
c     *                                                *
c     *  all distances and positions are measured      *
c     *  in terms of pixels                            *
c     *                                                *
c     **************************************************
c
c     **************************************************
c     *                                                *
c     *      Dennis Poulos                             *
c     *      x3403                                     *
c     *      22 April 1986                             *
c     *                                                *
c     **************************************************
      subroutine bmclear(bmd,x,y,w,h,color)
            external bmClearRegion
            integer bmd,x,y,w,h.color
      call bmClearRegion(bmd,x,y,w,h.color)
      return
      end
c
c    ***********************************************
c    *                                             *
c    *   compile as                                *
c    *   f77 -c bmclear.f -ltoolsf -lbmf -lvtf     *
c    *                                             *
c    ***********************************************
```

```
c    ************************************************
c    *                                       *
c    *              SET THICKNESS                 *
c    *                                       *
c    *  This is a fortran subroutine compiled      *
c    *  separately to set the thickness of lines    *
c    *  drawn onto the bitmap. The line thickness    *
c    *  is set in terms of x number of pixels       *
c    *  wide. This routine is intended to be        *
c    *  compiled inline in a LISP environment to     *
c    *  provide LISP graphics calls. The function    *
c    *  is called as (setthick bmd thickness)       *
c    *  It returns a "t" when complete. The          *
c    *  parameters passed to the fortran subroutine  *
c    *  from LISP are:                            *
c    *                                       *
c    *  bmd  =    bitmap descriptor number          *
c    *                                       *
c    *  thickness =    number of pixels wide        *
c    *              the line is desired to be     *
c    *                                       *
c    ************************************************
c
c    ************************************************
c    *                                       *
c    *      Dennis Poulos                       *
c    *      x 3403                          *
c    *      22 April 1986                      *
c    *                                       *
c    ************************************************
c
     subroutine setthick(bmd,thickness)
        integer bmd,thickness
        call bmSetThickness(bmd,thickness)
     return
     end
c
c    ************************************************
c    *                                       *
c    *  compile as                             *
c    *  f77 -c setthick.f -ltoolsf -lbmf -lvtf       *
c    *                                       *
c    ************************************************
```

```
c    ************************************************
c    *                                          *
c    *          SET X Y POSITION                       *
c    *                                          *
c    *  This is a fortran subroutine compiled          *
c    *  separately to position the position            *
c    *  pointer of a bitmap. This routine is           *
c    *  intended to be compiled inline in a LISP       *
c    *  environment to provide LISP graphics calls.    *
c    *  The function is called as                      *
c    *  (bmsetpos bmd x y) and it will return a        *
c    *  "t" when complete. The parameters passed       *
c    *  to the fortran subroutine from LISP are:       *
c    *                                          *
c    *  bmd  =   bitmap descriptor number              *
c    *                                          *
c    *  x    =   x coordinate of the pointer           *
c    *                                          *
c    *  y    =   y coordinate of the pointer           *
c    *                                          *
c    ************************************************
c
c    ************************************************
c    *                                          *
c    *      Dennis Poulos                       *
c    *      x 3403                              *
c    *      20 April 1986                       *
c    *                                          *
c    ************************************************
c
     subroutine bmsetpos(bmd,x,y)
        integer bmd,x,y
        call bmSetPosition(bmd,x,y)
     return
     end
c
c    ************************************************
c    *                                          *
c    *  compile as                              *
c    *  f77 -c bmsetpos.f -ltoolsf -lbmf -lvtf  *
c    *                                          *
c    ************************************************
```

```
c    **************************************************
c    *                                              *
c    *            SET LINE STYLE                     *
c    *                                              *
c    *  This is a fortran subroutine compiled        *
c    *  separately to set the style of the line      *
c    *  to be drawl on the bitmap. This consists     *
c    *  of a 16 bit mask as an integer number and    *
c    *  a count that is the number of times each     *
c    *  bit in the mask is replacated. This routine  *
c    *  is intended to be compiled inline in a       *
c    *  LISP environment to provide LISP graphics    *
c    *  calls. The function is called as             *
c    *  (bmsetsty bmd mask count) and it will        *
c    *  return a "t" when complete. The parameters   *
c    *  passed to the fortran subroutine from LISP   *
c    *  are:                                         *
c    *                                              *
c    *  bmd  =    bitmap descriptor number           *
c    *                                              *
c    *  mask =    bit mask for line                  *
c    *                                              *
c    *  count=    repetition count for bits in mask  *
c    *                                              *
c    **************************************************
c
c    **************************************************
c    *                                              *
c    *      Dennis Poulos                            *
c    *      x 3403                                   *
c    *      21 April 1986                            *
c    *                                              *
c    **************************************************
c
     subroutine bmsetsty(bmd.mask,count)
        integer bmd.mask,count
        call bmSetStyle(bmd,mask,count)
     return
     end
c
c    **************************************************
c    *                                              *
c    *  compile as                                   *
c    *  f77 -c bmsetsty.f -ltoolsf -lbmf -lvtf       *
c    *                                              *
c    **************************************************
```

```
c    ***************************************************
c    *                                               *
c    *              DRAW LINE                        *
c    *                                               *
c    *  This is a fortran subroutine compiled        *
c    *  separately to draw a line from the pointer   *
c    *  to the x and y position of the end of the    *
c    *  line. This routine is intended to be         *
c    *  compiled inline in a LISP environment to     *
c    *  provide LISP graphics calls. The function    *
c    *  is called as (bmdrawl bmd x y) and it will   *
c    *  return a "t" when complete. The parameters   *
c    *  passed to the fortran subroutine from LISP   *
c    *  are:                                         *
c    *                                               *
c    *  bmd  =   bitmap descriptor number            *
c    *                                               *
c    *  x    =   x coordinate of line end            *
c    *                                               *
c    *  y    =   y coordinate of line end            *
c    *                                               *
c    ***************************************************
c
c    ***************************************************
c    *                                               *
c    *      Dennis Poulos                            *
c    *      x 3403                                   *
c    *      20 April 1986                            *
c    *                                               *
c    ***************************************************
c
      subroutine bmdrawl(bmd,x,y)
         integer bmd,x,y
         call bmPaintLine(bmd,x,y)
      return
      end
c
c    ***************************************************
c    *                                               *
c    *  compile as                                   *
c    *  f77 -c bmdrawl.f -ltoolsf -lbmf -lvtf        *
c    *                                               *
c    ***************************************************
```

```
c    *************************************************
c    *                                               *
c    *         DISPLAY BIT MAP                       *
c    *                                               *
c    *  This is a fortran subroutine compiled        *
c    *  separately to display a bitmap in a window.  *
c    *  This routine is intended to be compiled      *
c    *  inline in a LISP environment to provide      *
c    *  LISP graphics calls. The function is called  *
c    *  "bmdisp". From LISP it is called as          *
c    *  (bmdisp fd bmd mode sx sy dx dy w h color)   *
c    *  and it will return 0 which equals the        *
c    *  copyraster function. The parameters passed   *
c    *  to the fortran function from LISP are:       *
c    *                                               *
c    *  fd   =   file descriptor number              *
c    *                                               *
c    *  bmd  =   bitmap descriptor number            *
c    *                                               *
c    *  type =   type of bmdisplay (copyraster       *
c    *           toggleraster, paintraster)          *
c    *                                               *
c    *  sx   =   x coordinate of upper left          *
c    *           corner of portion of bitmap         *
c    *                                               *
c    *  sy   =   y coordinate of upper left          *
c    *           corner of portion of bitmap         *
c    *                                               *
c    *  dx   =   x coordinate of upper left          *
c    *           corner of destination window        *
c    *                                               *
c    *  dy   =   y coordinate of upper left          *
c    *           corner of destination window        *
c    *                                               *
c    *  w    =   width of region (source and         *
c    *           display)                            *
c    *                                               *
c    *  h    =   height of region (source and        *
c    *           display)                            *
c    *                                               *
c    *  c    =   color (used if using paintraster,   *
c    *           otherwise set to 0)                 *
c    *                                               *
c    *************************************************
c
c    *************************************************
c    *                                               *
c    *      Dennis Poulos                            *
c    *      x 3403                                   *
c    *      19 April 1986                            *
c    *                                               *
c    *************************************************
```

```
c
      integer function bmdisp(fd,type,bmd,sx,sy,dx,dy,w,h,color)
         external bmDisplayBitmap
      integer bmDisplayBitmap,fd,type,bmd,sx,sy,dx,dy,w,h,color
      bmdisp = bmDisplayBitmap(fd,type,bmd,sx,sy,dx,dy,w,h,color)
      return
      end
c
c     ***********************************************
c     *                                             *
c     *  compile as                                 *
c     *  f77 -c bmdisp.f -ltoolsf -lbmf -lvtf       *
c     *                                             *
c     ***********************************************
```

```
c    ***************************************************
c    *                                    *
c    *          CLOSE-WINDOW                        *
c    *
c    *  This is a fortran subroutine compiled       *
c    *  separately to close a window on the ISI      *
c    *  Graphics Machines. It is intended to be      *
c    *  compiled inline in a LISP environment to     *
c    *  provide LISP graphics calls. The function    *
c    *  is called "clw". From LISP it is called      *
c    *  as (clw fd) and it will return a "t" when    *
c    *  complete. The parameters passed to the       *
c    *  fortran function from LISP are:            *
c    *                                    *
c    *  fd   =   file descriptor of the desired      *
c    *           window                         *
c    *                                    *
c    ***************************************************
c
c    ***************************************************
c    *                                    *
c    *       Dennis Poulos                    *
c    *       x 3403                         *
c    *       15 April 1986                     *
c    *                                    *
c    ***************************************************
c
     subroutine clw(fd)
        integer fd
     call CloseWindow(fd)
     return
     end
c
c    ***************************************************
c    *                                    *
c    *  compile as                          *
c    *  f77 -c closew.f -ltools -lbmf -lvtf          *
c    *                                    *
c    ***************************************************
```

```
c     *************************************************
c     *                                               *
c     *          BITMAP DEALLOCATE                    *
c     *                                               *
c     *   This is a fortran subroutine compiled       *
c     *   separately to deallocate space in memory    *
c     *   for a bitmap. A bit map is maintained in    *
c     *   memory and can be displayed in a window     *
c     *   as desired. This routine is intended to be  *
c     *   compiled inline in a LISP environment to    *
c     *   provide LISP graphics calls. The function   *
c     *   is called "bmdeal". From LISP it is called  *
c     *   as (bmdeal bmd) and it will return a "t"    *
c     *   when complete. The parameters passed to     *
c     *   the fortran subroutine from LISP are:       *
c     *                                               *
c     *   bmd  =    bitmap descriptor                 *
c     *                                               *
c     *************************************************
c
c     *************************************************
c     *                                               *
c     *        Dennis Poulos                          *
c     *        x 3403                                 *
c     *        19 April 1986                          *
c     *                                               *
c     *************************************************
c
      subroutine bmdeal(bmd)
         integer bmd
         call bmDeallocate(bmd)
      return
      end
c
c     *************************************************
c     *                                               *
c     *   compile as                                  *
c     *   f77 -c bmdeal.f -ltoolsf -lbmf -lvtf        *
c     *                                               *
c     *************************************************
```

# APPENDIX C – LSCAN

```
;    *******************************************************
;    *                                 *
;    *   This is a lisp program to load and execute      *
;    *   graphics on the ISI machines and to do image       *
;    *   processing.                             *
;    *                                 *
;    *******************************************************
;
;
;    *******************************************************
;    *                                 *
;    *              GRAPHICS ROUTINES               *
;    *                                 *
;    *   The following "cfasl" lisp functions compile      *
;    *   inline the ISI graphics routines. They use         *
;    *   fortran libraries in the ISI machines. The         *
;    *   first term after the "cfasl" is the file name     *
;    *   of the object code produced by the fortran         *
;    *   compiler. The second term is the entry point       *
;    *   into the subroutine. The third term is the name    *
;    *   given to the LISP function created by the          *
;    *   "cfasl". The forth term is type of routine it      *
;    *   is and the last term are the libraries the        *
;    *   compiler needs to link to.                 *
;    *                                 *
;    *******************************************************
;
(cfasl 'tofile.o '_tofile 'tofile nil "-lvt -lm -lbm")
(cfasl 'openw.o '_opw_ 'opw "integer-function" "-ltoolsf -lbmf -lvtf")
(cfasl 'setldisc.o '_slined_ 'slined "integer-function" "-ltoolsf -lbmf -lvtf")
(cfasl 'bitmal.o '_bitmapa_ 'bitmapa "integer-function" "-ltoolsf -lbmf -lvtf")
(cfasl 'bmadres.o '_bmadres_ 'bmadres "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'writfile.o '_writfile_ 'writfile "integer-function" "-ltoolsf -lbmf -lvtf")
(cfasl 'setbacc.o '_setbacc_ 'setbacc "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'setcol.o '_setcol_ 'setcol "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'bmclear.o '_bmclear_ 'bmclear "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'setthick.o '_setthick_ 'setthick "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'bmsetpos.o '_bmsetpos_ 'bmsetpos "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'bmsetsty.o '_bmsetsty_ 'bmsetsty "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'bmdrawl.o '_bmdrawl_ 'bmdrawl' "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'bmdisp.o '_bmdisp_ 'bmdisp "integer-function" "-ltoolsf -lbmf -lvtf")
(cfasl 'closew.o '_clw_ 'clw "subroutine" "-ltoolsf -lbmf -lvtf")
(cfasl 'bmdeal.o '_bmdeal_ 'bmdeal "subroutine" "-ltoolsf -lbmf -lvtf")
;
;    *******************************************************
;    *                                 *
;    *              CONSTANTS                 *
;   *                                                  *
;   *   The following are constants used throughout   *
;   *   the simulation.                                *
;   *                                                  *
;   ****************************************************
;
(setq wwidth  576)                  ;Window Width
(setq wheight 448)                  :Window Height
(setq filenum  40)                 :a file counter to create
                                   ;new files to copy the bit
                                   ;maps to
(setq filebase filenum)             ;base number for the files
                                   ;set to file number
(setq numruns   5)                  :the number of desired runs
(setq w1x      35)                 ;X coordinate of the upper
                                   ;left corner of window 1
(setq w1y      50)                 ;Y coordinate of the upper
                                   :left corner of window 1
(setq xsep     54)                 :lateral separation between
                                   ;windows
(setq ysep     52)                 :vertical separation between
                                   ;windows
(setq pie     3.14159)              ;pi
(setq slope_limit 2.0)              ;limit between flat terrain
                                   ;and a sloping terrain
(setq curv_threshold 0.05)             ;threshold for measurable
                                   ; curvature
(setq unsafe_slope_limit 30.0)        ;threshold between safe and
                                   ;unsafe slope in degrees
(setq x0init   287)                 ;terrain bitmap x0 coordinate
(setq y0init   133)                 ;terrain bitmap y0 coordinate
(setq x0cminit 77)                  ;classification map x0
                                   ;coordinate
(setq y0cminit 13)                  ;classification map y0
                                   ;coordinate
(setq peak_char 'P)                 ;peak symbol for
                                   ;classification map
(setq pit_char  'L)                 ;pit symbol for
                                   ;classification map
(setq ridge_char '^)                ;ridge symbol for
                                   ;classification map
(setq ravine_char 'v)               ;ravine symbol for
                                   ;classification map
(setq saddle_char 'S)               ;saddle symbol for
                                   ;classification map
(setq pass_char 'Y)                 ;pass symbol for the
                                   ;classification map
(setq flat_char '-)                ;flat terrain symbol for
                                   ;classification map
(setq safe_slope_char '/)            ;safe slope symbol for
                                   ;classification map
(setq unsafe_slope_char 'U)          ;unsafe slope symbol for
```

```
                                :classification map
(setq edge_char 'E)                  ;edge symbol for
                                :classification map
(setq numacross 29)                  :0->numacross terrain
                                :cells across
(setq numdown   29)                  :0->numdown terrain
                                :cells down
(setq bmdx     9)                  ;bitmap delta x per
                                ;terrain cell
(setq bmdy     3)                  ;bitmap delta y per
                                ;terrain cell
(setq windows nil)                  ;sets list of windows
                                ;equal to the nil set
(setq rclass_map nil)                  ;sets the list of
                                :real_terrain classification
                                ; map pixels to nil
(setq nclass_map nil)                  ;sets the list of
                                ;noisy_terrain classification
                                :map pixels to nil
(array real_terrain t (1+ numacross)(1+ numdown) 3)
                                :30 x 30 x 3 array of map
                                :x,y,z coordinates
(array noisy_terrain t (1+ numacross)(1+ numdown) 3)
                                :30 x 30 x 3 array of
                                ;noisy x,y,z map
                                :coordinates
(setq noise_limit 1)                  ;elevation data (z) is
                                ;subject to +/- the
                                ;noise_limit in
                                ;noise measured in pixels
(setq full_line 65535)                  ;mask for a full line
(setq VT_ABSOLUTE 1)                  ;addressing mode
(setq VT_RELATIVE 0)                  ;relative addressing mode
(setq COPYRASTER 0)                  ;bitmap display mode
(setq TWSDISC 5)                  :line discipline mode
(setq VT_BLACK 0)                  ;black color
(setq VT_WHITE 1)                  ;white color
;
;   ************************************************************
;   *                                          *
;   *       USER DEFINED LISP FUNCTIONS                 *
;   *                                          *
;   *   The following are the user defined functions      *
;   *   that do the actual work in this simulation.      *
;   *                                          *
;   ************************************************************
;
;   **   "copy_bm_tofile" copys the bitmaps of the terrain
;   **   and the symbolic maps to files so that they may be
;   **   printed of the laser printer
;
(defun copy_bm_tofile (bmd)
    (setq filenum (1- filenum))
    (tofile bmd (get_pname (concat 'figure. filenum))))
;
;   **   "def_window" creates a list of the open windows
;   **   and attaches attributes to each window
;
(defun def_window (winnum xcoord ycoord width height name)
    (putprop winnum xcoord 'xcoord)
    (putprop winnum ycoord 'ycoord)
    (putprop winnum width  'width)
    (putprop winnum height 'height)
    (putprop winnum name   'name)
    (setq windows (append1 windows winnum)))
;
;   **   "xtrfunc" defines the transfer function for
;   **   the terrain map x y z coordinates to be
;   **   mapped into the window x y coordinates
;
(defun xtrfunc (x y)
    (- x0init (- (* bmdx x)(* bmdx y))))
;
;   **   "ytrfunc" defines the transfer function for
;   **   the terrain map x y z coordinates to be
;   **   mapped into the window x y coordinates
;
(defun ytrfunc (x y z)
    (- (+ y0init (* bmdy x)(* bmdy y)) z))
;
;   **   "arrayx" recovers the x value of a terrain array
;   **   for a given pixel pi pj
;
(defun arrayx (pi pj arrayname)
    (arraycall t arrayname pi pj 0))
;
;   **   "arrayy" recovers the y value of a terrain array
;   **   for a given pixel pi pj
;
(defun arrayy (pi pj arrayname)
    (arraycall t arrayname pi pj 1))
;
;   **   "arrayz" recovers the z value of a terrain array
;   **   for a given pixel pi pj
;
(defun arrayz (pi pj arrayname)
    (arraycall t arrayname pi pj 2))
;
;   **   "paint_saddle" paints the character "S" on the
;   **   symbolic map to represent a saddle.
;
(defun paint_saddle (bmd cmx cmy)
    (bmsetpos bmd (+ 10 cmx) (+ 2 cmy))
    (bmdrawl bmd -7 0)
    (bmdrawl bmd 0 5)
    (bmdrawl bmd 7 0)
    (bmdrawl bmd 0 4)
    (bmdrawl bmd -7 0))
;
;   **  "paint_pass" paints the character "P" on the
;   **  symbolic map to represent a pass.
;
(defun paint_pass (bmd cmx cmy)
    (bmsetpos bmd (+ 3 cmx) (+ 11 cmy))
    (bmdrawl bmd 0 -9)
    (bmdrawl bmd 5 0)
    (bmdrawl bmd 2 2)
    (bmdrawl bmd 0 1)
    (bmdrawl bmd -2 2)
    (bmdrawl bmd -4 0))
;
;   **  "paint_safe_slope" paints the character / on the
;   **  symbolic map to represent a safe slope.
;
(defun paint_safe_slope (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 2) (+ cmy 11))
    (bmdrawl bmd 9 -9))
;
;   **  "paint_edge" paints the character "E" on the
;   **  symbolic map to represent an edge.
;
(defun paint_edge (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 10) (+ cmy 2))
    (bmdrawl bmd -7 0)
    (bmdrawl bmd 0 5)
    (bmdrawl bmd 4 0)
    (bmdrawl bmd -4 0)
    (bmdrawl bmd 0 4)
    (bmdrawl bmd 7 0))
;
;   **  "paint_unsafe_slope" paints the character "U" on the
;   **  symbolic map to represent an unsafe_slope.
;
(defun paint_unsafe_slope (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 3) (+ cmy 2))
    (bmdrawl bmd 0 6)
    (bmdrawl bmd 1 2)
    (bmdrawl bmd 2 1)
    (bmdrawl bmd 1 0)
    (bmdrawl bmd 2 -1)
    (bmdrawl bmd 1 -2)
    (bmdrawl bmd 0 -6))
;
;   **  "paint_peak" paints the character for a peak symbol
;   **  on the symbolic map to represent a peak.
;
```

```
(defun paint_peak (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 2) (+ cmy 6))
    (bmdrawl bmd 4 -4)
    (bmdrawl bmd 0 9)
    (bmdrawl bmd 0 -9)
    (bmdrawl bmd 4 4))
;
;    **   "paint_pit" paints the character for a pit symbol
;    **   on the symbolic map to represent a pit.
;
(defun paint_pit (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 2) (+ cmy 7))
    (bmdrawl bmd 4 4)
    (bmdrawl bmd 0 -9)
    (bmdrawl bmd 0 9)
    (bmdrawl bmd 4 -4))
;
;    **   "paint_ravine" paints the character "V" on the symbolic
;    **   map to represent a ravine on the symbolic map.
;
(defun paint_ravine (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 2) (+ cmy 2))
    (bmdrawl bmd 4 9)
    (bmdrawl bmd 1 0)
    (bmdrawl bmd 4 -9))
;
;    **   "paint_ridge" paints the symbol "^" on the symbolic map
;    **   to represent a ridge on the symbolic map.
;
(defun paint_ridge (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 2) (+ cmy 11))
    (bmdrawl bmd 4 -9)
    (bmdrawl bmd 1 0)
    (bmdrawl bmd 4 9))
;
;    **   "paint_level" paints the character "-" on the symbolic
;    **   map to represent a level terrain on the symbolic map.
;
(defun paint_flat (bmd cmx cmy)
    (bmsetpos bmd (+ cmx 3) (+ cmy 7))
    (bmdrawl bmd 7 0))
;
;    **   "make_real_terrain" is a function that fills the
;    **   array real_terrain with x,y,z coordinates for the
;    **   synthetic terrain used in this model. At this time
;    **   it does not create any elevation (z). This will be
;    **   added later.
;
(defun make_real_terrain ()
 (prog (x y z pi pj r1 r2 r3 r4 r5 counter)
    (setq x 0)
    (setq y 0)
(setq z 0)
(setq pi 0)
(setq pj 0)
(setq counter 0)
(setq r5 (+ 2 (random 5)))
    loop
(store (real_terrain pi pj 0) x)
(store (real_terrain pi pj 1) y)
(store (real_terrain pi pj 2) z)
(cond (( and (= pi numacross )(= pj numdown ))(go loop1))
     ((= pi numacross )
             (setq pi 0)
             (setq x 0)
             (setq pj (1+ pj))
             (setq y (1+ y))
             (go loop))
     (t  (setq pi (1+ pi))
         (setq x (1+ x))
         (go loop)))
     loop1
 (cond ((= counter r5)(return)))
 (setq pi 0)
 (setq pj 0)
 (setq r1 (random 31))
 (setq r2 (random 31))
 (setq r3 (quotient (float (+ 5 (random 11))) 10.0 ))
 (setq r4 (float (random 51)))
     loop2
 (setq z (arrayz pi pj 'real_terrain))
 (cond ((evenp counter)
           (setq z (+ z (fix (times r4 ( cos
                (times r3 2.0 pie(quotient (sqrt
                (add (float (expt (- r1 pi) 2))
                (float (expt (- r2 pj) 2)))) 30 ))))))))
      (t
           (setq z (+ z (fix (times r4 ( sin
                          (times r3 2.0 pie
                (quotient (sqrt (add (float
                          (expt (- r1 pi) 2))
                (float (expt (- r2 pj) 2)))) 30 )))))))))
 (store (real_terrain pi pj 2) z)
 (cond ((and (= pi numacross)(= pj numdown))
           (setq counter (1+ counter))
           (go loop1))
      ((= pi numacross)
       (setq pi 0)
       (setq pj (1+ pj))
       (go loop2))
      (t
       (setq pi (1+ pi))
       (go loop2)))))
;
```

```
;   **   "make_noisy_terrain" takes the array real_terrain
;   **   and adds a random noise to the elevation data. The
;   **   noise is +-- the constant noise_limit.
;
(defun make_noisy_terrain ()
 (prog (pi pj)
    (setq pi 0)
    (setq pj 0)
       loop
    (store (noisy_terrain pi pj 0)(arrayx pi pj 'real_terrain))
    (store (noisy_terrain pi pj 1)(arrayy pi pj 'real_terrain))
    (store (noisy_terrain pi pj 2)
        (+ (arrayz pi pj 'real_terrain)
          (- noise_limit (random (1+ (* 2 noise_limit))))))
    (cond (( and (= pi numacross)(= pj numdown))
        (return))
       ((= pi numacross)
        (setq pi 0)
        (setq pj (1+ pj))
        (go loop))
       (t
        (setq pi (1+ pi))
        (go loop)))))
;
;   **   "initialize" opens the required four windows
;   **   and attaches the fd (file descriptor) number
;   **   and the bmd (bitmap descriptor) number to the
;   **   windowlist as attributes of each window. It also
;   **   sets the line discipline of each window to the
;   **   graphics mode vice the standard TTY mode.
;
(defun initialize ()
 (prog (windowlist x)
    (setq windowlist windows)
       loop
    (setq x (car windowlist))
    (setq windowlist (cdr windowlist))
    (putprop x (opw (get x 'xcoord)
              (get x 'ycoord)
              (get x 'width)
              (get x 'height)
              (get x 'name))
            'fdnum)
    (putprop x (bitmapa (get x 'width)(get x 'height)) 'bmd)
    (bmadres (get x 'bmd) VT_ABSOLUTE)
    (setbacc (get x 'bmd) VT_WHITE)
    (setcol (get x 'bmd) VT_BLACK)
    (bmsetsty (get x 'bmd) full_line 2)
    (bmclear (get x 'bmd) 0 0 wwidth wheight VT_WHITE)
    (slined (get x 'fdnum) TWSDISC)
    (cond (( not( null windowlist)) (go loop)))))
;
```

```
;    **   "draw_terrain_base" draws the base on which the
;    **   synthetic terrain is overlaid
;
(defun draw_terrain_base (bmd)
 (prog (xl xm yu xr ym yl xr)
    (setq xm x0init)
    (setq yu y0init)
    (setq xr (+ x0init (* numacross bmdx)))
    (setq ym (+ y0init (* numdown bmdy )))
    (setq yl (+ y0init (* 2 numdown bmdy)))
    (setq xl (- x0init (* numacross bmdx)))
    (bmsetpos bmd xm yu)
    (setthick bmd 2)
    (bmdrawl bmd xr ym)
    (bmdrawl bmd xm yl)
    (bmdrawl bmd  xl ym)
    (bmdrawl bmd xm yu)
    (setthick bmd 1)))
;
;    **   "draw_terrain" takes the x y z coordinates of
;    **   the terrain held in the terrain arrays and
;    **   plots them
;
(defun draw_terrain (arrayname bmd fd)
 (prog (pi pj x y z bmx bmy)
    (setq pi 0)
    (setq pj 0)
    (draw_terrain_base bmd)
        loop
    (setq x (arrayx pi pj arrayname))
    (setq y (arrayy pi pj arrayname))
    (setq z (arrayz pi pj arrayname))
    (setq bmx (xtrfunc x y))
    (setq bmy (ytrfunc x y z))
    (bmsetpos bmd bmx bmy)
        loop1
    (cond((and(= pi numacross)(= pj numdown))
         (setthick bmd 2)
         (bmdrawl bmd bmx (+ bmy z))
         (bmsetpos bmd bmx bmy)
         (setthick bmd 1)
         (bmdisp fd COPYRASTER bmd 0 0 0 0 wwidth wheight 0)
         (return))
        ((= pj numdown)
         (setthick bmd 2)
         (bmdrawl bmd bmx (+ bmy z))
         (bmsetpos bmd bmx bmy)
         (setthick bmd 1)
         (setq pi (1+ pi))
         (setq x (arrayx pi pj arrayname))
         (setq y (arrayy pi pj arrayname))
         (setq z (arrayz pi pj arrayname))
```

```
     (setq bmx (xtrfunc x y))
     (setq bmy (ytrfunc x y z))
     (bmdrawl bmd bmx bmy)
     (go loop1))
    ((= pi numacross)
     (setthick bmd 2)
     (bmdrawl bmd bmx (+ bmy z))
     (bmsetpos bmd bmx bmy)
     (setthick bmd 1)
     (setq pj (1+ pj))
     (setq x (arrayx pi pj arrayname))
     (setq y (arrayy pi pj arrayname))
     (setq z (arrayz pi pj arrayname))
     (bmdrawl bmd (xtrfunc x y)(ytrfunc x y z))
     (setq pi 0)
     (go loop))
    ((or (= pj 0)(= pi 0))
     (setthick bmd 2)
     (bmdrawl bmd bmx (+ bmy z))
     (bmsetpos bmd bmx bmy)
     (setthick bmd 1)
     (setq pi (1+ pi))
     (setq x (arrayx (1- pi)(1+ pj) arrayname))
     (setq y (arrayy (1- pi)(1+ pj) arrayname))
     (setq z (arrayz (1- pi)(1+ pj) arrayname))
     (bmdrawl bmd (xtrfunc x y)(ytrfunc x y z))
     (bmsetpos bmd bmx bmy)
     (setq x (arrayx pi pj arrayname))
     (setq y (arrayy pi pj arrayname))
     (setq z (arrayz pi pj arrayname))
     (setq bmx (xtrfunc x y))
     (setq bmy (ytrfunc x y z))
     (bmdrawl bmd bmx bmy)
     (go loop1))
    (t
     (setq pi (1+ pi))
     (setq x (arrayx (1- pi)(1+ pj) arrayname))
     (setq y (arrayy (1- pi)(1+ pj) arrayname))
     (setq z (arrayz (1- pi)(1+ pj) arrayname))
     (bmdrawl bmd (xtrfunc x y)(ytrfunc x y z))
     (bmsetpos bmd bmx bmy)
     (setq x (arrayx pi pj arrayname))
     (setq y (arrayy pi pj arrayname))
     (setq z (arrayz pi pj arrayname))
     (setq bmx (xtrfunc x y))
     (setq bmy (ytrfunc x y z))
     (bmdrawl bmd bmx bmy)
     (go loop1)))))
;
;  **  "draw_class_map_base" draws the base in which the
;  **  classification map is displayed
;
```

```
(defun draw_class_map_base (bmd)
 (prog ()
    (setthick bmd 2)
    (bmsetpos bmd (- x0cminit 5) (- y0cminit 5))
    (bmadres bmd VT_RELATIVE)
    (bmdrawl bmd 430 0)
    (bmdrawl bmd 0 430)
    (bmdrawl bmd -430 0)
    (bmdrawl bmd 0 -430)
    (setthick bmd 1)))
;
;    **   "make_rclass_map" builds the list of
;    **   pixels associated with a classification
;    **   map. It will determine if a pixel needs to be
;    **   classified or if it is a edge pixel.
;    **   If it needs to be classified it will
;    **   invoke the function "classify".
;
(defun make_rclass_map ()
 (prog (pi pj x y z )
    (setq pi 0)
    (setq pj 0)
    (setq arrayname 'real_terrain)
        loop
    (setq pixname (concat 'pixel- (+ pi (* 30 pj))))
    (putprop pixname (arrayx pi pj arrayname) 'x)
    (putprop pixname (arrayy pi pj arrayname) 'y)
    (putprop pixname (quotient (float (arrayz pi pj arrayname)
                                ) 10.0) 'z)
    (cond ((or (= pi 0)(= pi numacross)(= pj 0)(= pj numdown))
          (putprop pixname 'edge 'pri_class)
          (putprop pixname edge_char 'letter))
         (t
          (classify pixname arrayname pi pj)))
    (setf rclass_map (append1 rclass_map pixname))
    (cond ((and (= pi numacross)(= pj numdown))
          (return))
         ((= pi numacross)
          (setq pi 0)
          (setq pj (1+ pj))
          (go loop))
         (t
          (setq pi (1+ pi))
          (go loop)))))
;
;    **   "make_nclass_map" builds the list of pixels associated
;    **   with a classification map. It will determine if a
;    **   pixel needs to be classified or if it is a edge pixel.
;    **   If it needs to be classified it will invoke the
;    **   function "classify".
;
(defun make_nclass_map ()
 (prog (pi pj x y z )
    (setq pi 0)
    (setq pj 0)
    (setq arrayname 'noisy_terrain)
        loop
    (setq pixname (concat 'pixel- (+ pi (* 30 pj))))
    (putprop pixname (arrayx pi pj arrayname) 'x)
    (putprop pixname (arrayy pi pj arrayname) 'y)
    (putprop pixname (quotient (float (arrayz pi pj arrayname)
                                ) 10.0) 'z)
    (cond ((or (= pi 0)(= pi numacross)(= pj 0)(= pj numdown))
          (putprop pixname 'edge 'pri_class)
          (putprop pixname edge_char 'letter))
         (t
          (classify pixname arrayname pi pj)))
    (setf nclass_map (append1 nclass_map pixname))
    (cond ((and (= pi numacross)(= pj numdown))
          (return))
         ((= pi numacross)
          (setq pi 0)
          (setq pj (1+ pj))
          (go loop))
         (t
          (setq pi (1+ pi))
          (go loop)))))
;
;   **   "classify" does the real work of generating the
;   **   max and min curvature and the slope of each
;   **   pixel and deciding what it means.
;
(defun classify (pixname arrayname pi pj)
 (prog (i j x y z k2 k3 k4 k5 k6 elev b c E1 E2 temp slope)
    (setq i (1- pi))
    (setq j (1- pj))
    (setq x 0)
    (setq y 0)
    (setq z 0)
    (setq k2 0)
    (setq k3 0)
    (setq k4 0)
    (setq k5 0)
    (setq k6 0)
    (setq elev 0)
        loop1
    (setq x (diff (float (arrayx i j arrayname))
                    (float (arrayx pi pj arrayname))))
    (setq y (diff (float (arrayy i j arrayname))
                    (float (arrayy pi pj arrayname))))
    (setq z (quotient
           (diff (float (arrayz i j arrayname))
                 (float (arrayz pi pj arrayname)))
           10.0))
    (setq k2 (plus k2 (times x z)))
    (setq k3 (plus k3 (times y z)))
    (setq k4 (plus k4 (times x x z)))
    (setq k5 (plus k5 (times x y z)))
    (setq k6 (plus k6 (times y y z)))
    (setq elev (plus elev z))
    (cond ((= i (1+ pi))
          (setq i (1- pi))
          (setq j (1+ j))
          (go loop1))
         ((and (nequal i (1+ pi)) (nequal j (1+ pj)))
          (setq i (1+ i))
          (go loop1)))
    (setq k2 (quotient k2 6.0))
    (setq k3 (quotient k3 6.0))
    (setq k4 (diff (quotient k4 2.0)
                  (quotient elev 3.0)))
    (setq k5 (quotient k5 4.0))
    (setq k6 (diff (quotient k6 2.0)
                  (quotient elev 3.0)))
    (setq b (minus (sum (times 2.0 k4)(times 2.0 k6))))
    (setq c (diff (times 4.0 k4 k6)(times k5 k5)))
    (setq E1 (quotient (sum (minus b)
                      (sqrt (diff (times b b)(times 4.0 c))))
                       2.0))
    (setq E2 (quotient (diff (minus b)
                      (sqrt (diff (times b b)(times 4.0 c))))
                       2.0))
    (cond ((greaterp (abs E2)(abs E1))
          (setq temp E1)
          (setq E1 E2)
          (setq E2 temp)))
    (setq slope (quotient (times 360.0
                        (atan (sqrt (sum (times k2 k2)
                                        (times k3 k3))) 1.0))
                2.0 pie))
    (putprop pixname slope 'slope)
    (putprop pixname E1 'max_curvature)
    (putprop pixname E2 'min_curvature)
;FIRST RULE
;- IF SLOPE IS APPROX 0 - THE PRIMARY CLASSIFICATION
;OF THE PIXEL IS LEVEL TERRAIN
    (cond ((< slope slope_limit)
          (putprop pixname 'level 'pri_class)
;-AND IF MAX AND MIN CURVATURE IS NEGATIVE
;-PEAK IS SECONDARY CLASSIFICATION
;-PEAK IS 'LETTER
          (cond ((and (< E1 (minus curv_threshold))
                     (< E2 (minus curv_threshold)))
                (putprop pixname 'peak 'sec_class)
                (putprop pixname peak_char 'letter))
;OR MAX AND MIN CURVATURE IS POSITIVE
;-PIT IS SECONDARY CLASSIFICATION
;-PIT IS 'LETTER
             ((and (> E1 curv_threshold)
                  (> E2 curv_threshold))
              (putprop pixname 'pit 'sec_class)
              (putprop pixname pit_char 'letter))
;OR MAX CURVATURE IS NEGATIVE WHILE MIN CURVATURE IS APPROX 0
;-RIDGE IS SECONDARY CLASSIFICATION
;-RIDGE IS 'LETTER
             ((and (< E1 (minus curv_threshold))
                  (< (abs E2) curv_threshold))
              (putprop pixname 'ridge 'sec_class)
              (putprop pixname ridge_char 'letter))
;OR MAX CURVATURE IS POSITIVE WHILE MIN CURVATURE IS APPROX 0
;-RAVINE IS SECONDARY CLASSIFICATION
;-RAVINE IS 'LETTER
             ((and (> E1 curv_threshold)
                  (< (abs E2) curv_threshold))
              (putprop pixname 'ravine 'sec_class)
              (putprop pixname ravine_char 'letter))
;OR IF THE MAX CURVATURE IS NEGATIVE WHILE THE MIN CURVATURE
;POSITIVE
;-SADDLE IS SECONDARY CLASSIFICATION
;-SADDLE IS 'LETTER
             ((and (< E1 (minus curv_threshold))
                  (> E2 curv_threshold))
              (putprop pixname 'saddle 'sec_class)
              (putprop pixname saddle_char 'letter))
;OR IF THE MAX CURVATURE IS POSITIVE WHILE THE MIN CURVATURE
;IS NEGATIVE
;-PASS IS SECONDARY CLASSIFICATION
;-PASS IS 'LETTER
             ((and (> E1 curv_threshold)
                  (< E2 (minus curv_threshold)))
              (putprop pixname 'pass 'sec_class)
              (putprop pixname pass_char 'letter))
;EVERYTHING ELSE IS
;-FLAT TERRAIN IS SECONDARY CLASSIFICATION
;-FLAT IS 'LETTER
             (t
              (putprop pixname 'flat 'sec_class)
              (putprop pixname flat_char 'letter))))
;- ELSE, IF THE SLOPE IS LESS THAN THE UNSAFE_SLOPE LIMIT THEN
;THE PRIMARY CLASSIFICATION IS SAFE SLOPE
;-AND SAFE SLOPE IS THE 'LETTER
         ((< slope unsafe_slope_limit)
          (putprop pixname safe_slope_char 'letter)
          (putprop pixname 'safe_slope 'pri_class))
;- ELSE EVERYTHING ELSE IS UNSAFE_SLOPE PRIMARY CLASSIFICATION
;-AND UNSAFE SLOPE IS THE 'LETTER
         (t
          (putprop pixname 'unsafe_slope 'pri_class)
```

```
        (putprop pixname unsafe_slope_char 'letter))
;AND MAX AND MIN CURVATURE IS NEGATIVE
;-PEAK IS SECONDARY CLASSIFICATION
       (cond ((and (< E1 (minus curv_threshold))
                   (< E2 (minus curv_threshold)))
             (putprop pixname 'peak 'sec_class))
;OR MAX AND MIN CURVATURE IS POSITIVE
;-PIT IS SECONDARY CLASSIFICATION
            ((and (> E1 curv_threshold)
                  (> E2 curv_threshold))
             (putprop pixname 'pit 'sec_class))
;OR MAX CURVATURE IS NEGATIVE WHILE MIN CURVATURE IS APPROX 0
;-RIDGE IS SECONDARY CLASSIFICATION
            ((and (< E1 (minus curv_threshold))
                  (< (abs E2) curv_threshold))
             (putprop pixname 'ridge 'sec_class))
;OR MAX CURVATURE IS POSITIVE WHILE MIN CURVATURE IS APPROX 0
;-RAVINE IS SECONDARY CLASSIFICATION
            ((and (> E1 curv_threshold)
                  (< (abs E2) curv_threshold))
             (putprop pixname 'ravine 'sec_class))
;OR IF THE MAX CURVATURE IS NEGATIVE WHILE THE MIN CURVATURE
;POSITIVE
;-SADDLE IS SECONDARY CLASSIFICATION
            ((and (< E1 (minus curv_threshold))
                  (> E2 curv_threshold))
             (putprop pixname 'saddle 'sec_class))
;OR IF THE MAX CURVATURE IS POSITIVE WHILE THE MIN CURVATURE
;IS NEGATIVE
;-PASS IS SECONDARY CLASSIFICATION
            ((and (> E1 curv_threshold)
                  (< E2 (minus curv_threshold)))
             (putprop pixname 'pass 'sec_class))
;EVERYTHING ELSE IS
;-FLAT TERRAIN IS SECONDARY CLASSIFICATION
            (t
             (putprop pixname 'flat 'sec_class)))))
;
;   **  "draw_rclass_map" is the function to call to create
;   **  a real classification map, draw the class map base
;   **  to a bit map, and then draw the classification
;   **  map to the bit map.
;
(defun draw_rclass_map (bmd fd)
 (prog (pi pj map x cmx cmy)
    (make_rclass_map)
    (draw_class_map_base bmd)
    (setq pi 0)
    (setq pj 0)
    (setq map rclass_map)
        loop
    (setq x (car map))
    (setq map (cdr map))
    (setq cmx (- x0cminit (* 14 pi)))
    (setq cmy (- y0cminit (* 14 pj)))
    (cond ((= (get x 'letter) '/)
           (paint_safe_slope bmd cmx cmy))
          ((= (get x 'letter) 'U)
           (paint_unsafe_slope bmd cmx cmy))
          ((= (get x 'letter) 'E)
           (paint_edge bmd cmx cmy))
          ((= (get x 'letter) '-)
           (paint_flat bmd cmx cmy))
          ((= (get x 'letter) 'P)
           (paint_peak bmd cmx cmy))
          ((= (get x 'letter) 'L)
           (paint_pit bmd cmx cmy))
          ((= (get x 'letter) '^)
           (paint_ridge bmd cmx cmy))
          ((= (get x 'letter) 'v)
           (paint_ravine bmd cmx cmy))
          ((= (get x 'letter) 'Y)
           (paint_pass bmd cmx cmy))
          ((= (get x 'letter) 'S)
           (paint_saddle bmd cmx cmy)))
    (cond ((and (= pi numacross)
                (= pj numdown))
           (bmdisp fd COPYRASTER bmd 0 0 0 0 wwidth wheight 0)
           (bmadres bmd VT_ABSOLUTE)
           (return))
          ((= pi numacross)
           (setq pi 0)
           (setq pj (1+ pj))
           (go loop))
          (t
           (setq pi (1+ pi))
           (go loop)))))
;
;   **  "draw_nclass_map" is the function to call
;   **  to create a noisy classification map,
;   **  draw the class map base to a bit map, and then
;   **  draw the classification map to the bit map.
;
(defun draw_nclass_map (bmd fd)
 (prog (pi pj map x cmx cmy)
    (make_nclass_map)
    (draw_class_map_base bmd)
    (setq pi 0)
    (setq pj 0)
    (setq map nclass_map)
       loop
    (setq x (car map))
    (setq map (cdr map))
    (setq cmx (+ x0cminit (* 14 pi)))
    (setq cmy (- y0cminit (* 14 pj)))
    (cond ((= (get x 'letter) '/)
           (paint_safe_slope bmd cmx cmy))
          ((= (get x 'letter) 'U)
           (paint_unsafe_slope bmd cmx cmy))
          ((= (get x 'letter) 'E)
           (paint_edge bmd cmx cmy))
          ((= (get x 'letter) '-)
           (paint_flat bmd cmx cmy))
          ((= (get x 'letter) 'P)
           (paint_peak bmd cmx cmy))
          ((= (get x 'letter) 'L)
           (paint_pit bmd cmx cmy))
          ((= (get x 'letter) '^)
           (paint_ridge bmd cmx cmy))
          ((= (get x 'letter) 'v)
           (paint_ravine bmd cmx cmy))
          ((= (get x 'letter) 'Y)
           (paint_pass bmd cmx cmy))
          ((= (get x 'letter) 'S)
           (paint_saddle bmd cmx cmy)))
    (cond ((and (= pi numacross)
                (= pj numdown))
           (bmdisp fd COPYRASTER bmd 0 0 0 0 wwidth wheight 0)
           (bmadres bmd VT_ABSOLUTE)
           (return))
          ((= pi numacross)
           (setq pi 0)
           (setq pj (1+ pj))
           (go loop))
          (t
           (setq pi (1+ pi))
           (go loop)))))
;
;   **  "draw1" draws the first display, that being
;   **  the real terrain
;
(defun draw1 ()
    (initialize)
    (make_real_terrain)
    (draw_terrain 'real_terrain (get(car windows)'bmd)
                                (get(car windows)'fdnum))
    (copy_bm_tofile (get (car windows) 'bmd))
    (make_noisy_terrain)
    (draw_terrain 'noisy_terrain ( get (cadr windows) 'bmd)
                                 ( get (cadr windows) 'fdnum))
    (copy_bm_tofile (get (cadr windows) 'bmd))
    (draw_rclass_map (get (caddr windows) 'bmd)
                     (get (caddr windows) 'fdnum))
    (copy_bm_tofile (get (caddr windows) 'bmd))
    (draw_nclass_map (get (cadddr windows) 'bmd)
                     (get (cadddr windows) 'fdnum))
    (copy_bm_tofile (get (cadddr windows) 'bmd)))
;
;   **   "redraw" redraws a new real terrain into window 1
;
(defun redraw ()
 (prog (bm fd)
    (setq bm (get (car windows) 'bmd))
    (setq fd (get (car windows) 'fdnum))
    (bmclear bm 0 0 wwidth wheight VT_WHITE)
    (make_real_terrain)
    (draw_terrain 'real_terrain bm fd)
    (copy_bm_tofile bm)
    (setq bm (get (cadr windows) 'bmd))
    (setq fd (get (cadr windows) 'fdnum))
    (bmclear bm 0 0 wwidth wheight VT_WHITE)
    (make_noisy_terrain)
    (draw_terrain 'noisy_terrain bm fd)
    (copy_bm_tofile bm)
    (setq bm (get (caddr windows) 'bmd))
    (setq fd (get (caddr windows) 'fdnum))
    (bmclear bm 0 0 wwidth wheight VT_WHITE)
    (draw_rclass_map bm fd)
    (copy_bm_tofile bm)
    (setq bm (get (cadddr windows) 'bmd))
    (setq fd (get (cadddr windows) 'fdnum))
    (bmclear bm 0 0 wwidth wheight VT_WHITE)
    (draw_nclass_map bm fd)
    (copy_bm_tofile bm)))
;
;   **   "keep_running" starts the program
;   **   with a call to "draw1" then it checks
;   **   to see if it has completed the required
;   **   number of runs. If it has it closes all
;   **   the windows and exits lisp. If not it
;   **   keeps running.
;
(defun keep_running ()
 (prog ()
   (draw1)
       loop10
   (cond ((= filenum (+ (* 4 numruns) filebase))
        (finish)
        (exit))
       (t
        (redraw)
        (go loop10)))))
;
;   **   "finish" closes all the open windows and
;   **   deallocates the bitmap memory space.
;
(defun finish ()
 (prog (windowlist x)
    (setq windowlist windows)
        loop
    (setq x (car windowlist))
    (setq windowlist (cdr windowlist))
    (clw (get x 'fdnum))
    (bmdeal (get x 'bmd))
    (cond (( not (null windowlist)) (go loop)))))
;
;   *****************************************************
;   *                                                   *
;   *          THE MAIN PROGRAM                         *
;   *                                                   *
;   *****************************************************
;
;   **   Define window one
;
(def_window 'w1 w1x w1y
    wwidth wheight "TERRAIN")
;
;   **   Define window two
;
(def_window 'w2 w1x (plus w1y ysep wheight)
    wwidth wheight "NOISY TERRAIN")
;
;   **   Define window three
;
(def_window 'w3 (plus w1x xsep wwidth) w1y
    wwidth wheight "CLASSIFICATION MAP")
;
;   **   Define window four
;
(def_window 'w4 (plus w1x xsep wwidth) (plus w1y ysep wheight)
    wwidth wheight "NOISY CLASSIFICATION MAP")
;
;   **   Start the simulation
;
(keep_running)
;
;   ************************************************
;   *                                              *
;   *            THE END                           *
;   *                                              *
;   ************************************************
;
```

## LIST OF REFERENCES

1. Brady. Michael. "Artificial Intelligence and Robotics." Artificial Intelligence. v. 26. 1985.

2. Hecht-Nielson. R.. "Neural Analog Processing." Proceeding of SPIE . v. 360. 1982.

3. Nitzan. D., and others. "The Measurement and Use of Registered Reflectance and Range Data in Scene Analysis," Proc. IEEE , v. 65, no. 2, February 1977.

4. Nitzan. D., "The Development of Intelligent Robots: Achievements and Issues." IEEE Journal of Robotics and Automation , v. 1, no. 1, March 1985.

5. Zuk, D., and others. A System for Autonomous Land Navigation , paper presented at the Active Systems Workshop, Naval Postgraduate School. Monterey, California. November 1985.

6. Nathanson. Fred E., Radar Design Principles , McGraw-Hill Book Company. 1969.

7. Hecht. E.. and Zajac. A.. Optics , Addison-Wesley Publishing Company. 1974.

8. Fuhs. Alan E., High Energy Laser System Design , unpublished class notes, Naval Postgraduate School, 1985.

9. Charniak. E. and McDermott. D., Introduction to Artificial Intelligence , Addison-Wesley Publishing Company, 1985.

10. Nevatia. R.. Machine Perception , Prentice-Hall, Inc., 1982.

11. Moravec. Hans P., Robot Rover Visual Navigation , UMI Research Press. 1981.

12. Ozguner. F., Tsai, S. J., and McGhee, R. B., "An Approach to the Use of Terrain-Preview Information in Rough-Terrain Locomotion by a Hexapod

Walking Machine." The International Journal Of Robotics Research . v. 3. no. 2. Summer 1984.

13. Rensselaer Polytechnic Institute UNCLASSIFIED letter to Defense Advanced Research Projects Agency. Subject: RPI Proposal No. 322(12R)2306(36H) entitled Vision Systems for Mobile Robots , 6 October 1981.

14. McFarland. William D. and McLaren. Robert W., "Problems in Three-dimensional Imaging," Proceedings of SPIE Intelligent Robots: Third International Conference on Robot Vision and Sensory Controls , v. 449, 7-10 November 1983.

15. McFarland. William D., "Three-Dimensional Images for Robot Vision." Proceedings of SPIE Robotics and Robot Sensing Systems , v. 442, 25 August 1983.

16. O'Shea. Donald C., Callen. W. Russell, and Rhodes. William T., Introduction to Lasers and Their Applications , Addison-Wesley Publishing Company, 1978.

17. Tsai, Sheng-Jen, An Experimental Study of a Binocular Vision System for Rough Terrain Locomotion of a Hexapod Walking Robot , Ph. D. Thesis. Ohio State University, Columbus, Ohio, 43210, 1983.

18. Jorgensen, C., Hammel, W., and Weisben, C., "Autonomous Robot Navigation." BYTE The Small Systems Journal , v. 11, no. 1, January 1986.

19. Ittner, David J. and Jain, Anil K., "3-D Surface Discrimination From Local Curvature Measures." Proceedings of IEEE Computer Vision and Pattern Recoginition Conference , San Francisco, California, June 1985.

20. Marr, D., "Representing Visual Information - a Computational Approach." Computer Vision Systems . pp. 61-80. Academic Press, 1978.

21. Barrow, H. G. and Tenenbaum, J. M., "Recovering Intrinsic Scene Characteristics From Images." Computer Vision Systems , pp. 3-26, Academic Press, 1978.

22. Magee. Michael J., and others "Experiments in Intensity Guided Range Sensing Recognition of Three-Dimensional Objects." IEEE Transactions on

Pattern Analysis and Machine Intelligence . vol. PAMI-7. no. 6. November 1985.

23. Olivier. J. L. and Ozguner. F.. "A Navigation Algorithm for an Intelligent Vehicle with a Laser Rangefinder." Proceedings 1986 IEEE International Conference on Robotics and Automation ,v. 2. April 7-10. 1986.

24. Olivier. James L.. A Navigation Algorithm for an Intelligent Vehicle with a Laser Range Finder , Master's Thesis. Ohio State University. Columbus. Ohio. 43210. 1985.

25. Agrawal. Sunil Kumar, Mobility Characterization of Robotic Platforms . M. S. Thesis. Ohio State University. Columbus, Ohio 43210, Autumn 1985.

26. Anon.. The Autonomous Land Vehicle Program , Martin Marietta Denver Aerospace. Colorado. December 1985.

27. Brigham, Oran E., The Fast Fourier Transform , Prentice-Hall. Inc.. 1974.

28. Lee, Wha-Joon, A Computer Simulation Study of Omnidirectional Supervisory Control for Rough-terrain Locomotion by a Multilegged Robot Vehicle ,Ph.D. Thesis. Ohio State University. Columbus. Ohio, 43210, March 1984.

29. Thomas, George B. Jr. and Finney. Ross L., Calculus and Analytic Geometry , Addison-Wesley Publishing Company, 1982.

30. Brogan. William L., Modern Control Theory , Quantum Publishers. Inc., 1974.

31. McGhee. Robert B., Geometry of Quadratic Surfaces , class notes for CS 4800 Advanced Robotic Systems class at the Naval Postgraduate School, Monterey. California , 16 May 1986.

32. Wilensky, Robert, LISPcraft . W. W. Norton & Company. 1984.

33. Horgan, John, "Roboticists Aim to Ape Nature," IEEE Spectrum , v. 23, no. 2, February 1986.

34. Foderaro. John K.. The Franz LISP Manual . Regents of the University of California. 1980.

35. Integrated Solutions. The Programmer's Reference Manual for Graphics Software , Integrated Solutions, San Jose. California. September 1985.

36. Integrated Solutions. Libtoolsf: A Fortran Interface to Grpahics Tools , Integrated Solutions, San Jose, California, September 1985.

37. Integrated Solutions. Libvtf: A Fortran Interface to the Virtual Terminal , Integrated Solutions, San Jose, California, September 1985.

38. Integrated Solutions. Libbmf: A Fortran Interface to Bitmap Graphics . Integrated Solutions, San Jose, California, September 1985.

## INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria. Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943-5002 | 2 |
| 3. | Dr. Robert B. McGhee, Code 52Mz<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 20 |
| 4. | Dr. Larry W. Abbott. Code 62At<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 2 |
| 5. | Mr. Russell Davis<br>HQ, USACDEC<br>Attention: ATEC-IM<br>Fort Ord, California 93941 | 1 |
| 6. | Lieutenant Commander Dennis Poulos<br>FLECOMPRON FIVE<br>Box 39<br>FPO, San Francisco. Ca. 96654-2700 | 2 |
| 7. | Mr. and Mrs. Poulos<br>3728 Duffy Way<br>Bonita, California 92002 | 1 |

265

218815

Thesis
P75285 Poulos
c.1 Range image processing for local navigation of an autonomous land vehicle.