

Low Computation Vision-Based Navigation For Mobile Robots

By

Andrew S. Gavin

B.A. Haverford College (1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science


at the

Massachusetts Institute of Technology

May 1994

© Andrew S. Gavin, 1994

The author hereby grants to MIT permission to reproduce and to distribute copies of this thesis document in whole or in part.



Signature of Author.....

Department of Electrical Engineering and Computer Science
Wednesday, May 11, 1994

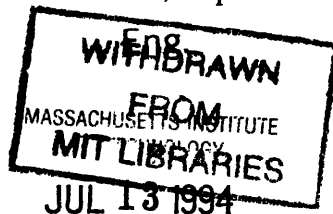
Certified by.....

Rodney A. Brooks
Professor of Computer Science
Thesis Supervisor



Accepted by.....

Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students



LIBRARIES

Low Computation Vision-Based Navigation For Mobile Robots

By
Andrew S. Gavin

Submitted to the Department of Electrical Engineering and Computer Science on Wednesday, May 11, 1994,
in partial fulfillment of the requirements for the degree of Scientific Masters

Abstract

In the design and construction of mobile robots vision has always been one of the most potentially useful sensory systems. In practice however, it has also become the most difficult to successfully implement. I have designed a small, light, cheap, and low power Mobot Vision System that can be used to guide a mobile robot in a constrained environment. The target environment is the surface of Mars, although I believe the system should be applicable to other conditions as well. It is my belief that the constraints of the Martian environment will allow the implementation of a system that provides vision based guidance to a small mobile rover.

The purpose of this vision system is to process realtime visual input and provide as output information about the relative location of safe and unsafe areas for the robot to go. It could also be programmed to do additional tasks such as tracking a small number of interesting features, for example the lander or large rocks (for scientific sampling). The system I have built was designed to be self contained. It has its own camera and on board processing unit. It draws a small amount of power and exchanges a very small amount of information with the host robot. The project had two parts, first the construction of a hardware platform, and second the implementation of a successful vision algorithm.

For the first part of the project I built a small self contained vision system. It employs a cheap but fast general purpose microcontroller (a 68332) connected to a Charge Coupled Device (CCD). The CCD provides the CPU with a continuous series of medium resolution gray-scale images (64 by 48 pixels with 256 gray levels at 10-30 frames a second). In order to accommodate my goals of low power, light weight, and small size I bypassed the traditional NTSC video and used a purely digital solution. As the frames are captured any desired algorithm can then be implemented on the microcontroller to extract the desired information from the images and communicate it to the host robot. Additionally, conventional optics are typically oversized for this application so I have employed aspheric lenses, pinhole lenses, and lens sets.

It is my hypothesis that a simple vision algorithm does not require huge amounts of computation and that goals such as constructing a complete three dimensional map of the environment are difficult, wasteful, and possibly unreachable. I believe that the nature of the environment can provide enough constraints to allow us to extract the desired information with a minimum of computation. It is also my belief that biological systems reflect an advanced form of this. They also employ constant factors in the environment to extract what information is relevant to the organism.

I believe that it is possible to construct a useful real world outdoor vision system with a small computational engine. This will be made feasible by an understanding of what information is desirable to extract from the environment for a given task, and of an analysis of the constraints imposed by the environment. In order to verify this hypothesis and to facilitate vision experiments I have built a small wheeled robot named Gopher, equipped with one of our vision systems. This robot has been run successfully in a number of environments, included a simulated Mars I have constructed for this purpose.

Thesis Supervisor: Rodney A. Brooks
Title: Professor of Computer Science

Acknowledgements

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research was provided by Jet Propulsion Laboratory contract number 959333. Additional support was provided by the Advanced Research Projects Agency under Office of Naval Research contract N00014-91-J-4038.

I would like to specially thank my advisor Rod Brooks, as well Masaki Yamamoto for his absolutely invaluable help designing, building, debugging, and supporting the hardware described here. Ian Horswill for his patience toward a new grad student with so many questions, as well as his influential ideas and for providing the code for some of the visual operators. Dave Miller for advice on Mars and long lunch conversations, Brian Wilcox and the other members of the JPL rover team, Anne Wright and Randy Sargent for their invaluable 68332 tools, Phil Alvelda, Olaf Bleck, Mark Torrance, and Mike Connell for their help.

I would also like to thank the other members of the Mobot Lab for countless help and advice, Anita Flynn, Cynthia Ferrell, Matthew Marjanovic, Maja Mataric, Mike Binnard, James McLurkin, and Eleni Kapogiannis. In addition are my fellow grad students and friends, Dave Baggett, Holly Yanco, Oded Maron, Dave Beymer, Mike Wessler, Greg Galperin, Carl de Marcken, Jason Rubin, Catherine Young, and the elusive How to Solve AI.

Finally, I would like to thank Mom, Dad, Mitch, Zeke, and Sol, without whom nothing in my life would be possible.

1	Introduction.	1
1.1	Mobile Robots.	1
1.2	Robot Vision.	3
1.3	Philosophy of Low Cost Vision.	5
1.4	Project Goals.	5
1.5	Roadmap.	6
2	Related Work.	7
2.1	Summary of Traditional Vision.	7
2.2	Low Computation Vision in Insects.	8
2.3	Summary of Primate Vision.	10
2.4	Previous Simple Vision Work at MIT.	13
3	Mars	17
3.1	Features of the Martian Environment.	18
3.2	The Mesur/Pathfinder Mission.	21
3.3	Constraints of the Martian Environment.	21
4	The Robot Vision System.	23
4.1	Image Size, Frame Rate, and Processor Speed.	23
4.2	Grabbing Frames: Digital Cameras.	26
4.3	The Camera Board.	27
4.4	The Camera Head.	28
4.5	The Frame Grabber Software.	29
4.6	The Debugging Board.	30
4.7	Cost Analysis of the Hardware System.	33
4.8	Gopher: A Vision Based Mobile Robot.	33
4.9	Sim Mars: A Simple Artificial Environment.	35
5	Software.	37
5.1	Images: How good is 64 by 48.	37
5.2	Low Level Vision Programming.	39
5.3	Software System.	49
5.4	Computational Costs of Simple Vision.	50
5.5	Applications to Mars.	52
6	Results.	60
7	Conclusions.	65
7.1	Summary.	65
7.2	Areas for Further Research.	66

Chapter 1

Introduction

In the design and construction of mobile robots vision has always been one of the most potentially useful sensory systems. In practice however, it has also become the most difficult to successfully implement. For my research I have designed a small, light, cheap, and low power Mobot Vision System that can be used to guide a mobile robot in a constrained environment. The target environment is the surface of Mars, although I believe the system should be applicable to other conditions as well. It is my belief that the constraints of the Martian environment allow the implementation of a system that provides vision based guidance to a small mobile rover.

1.1 Mobile Robots

The construction of viable mobile robots that imitate even simple animal competence has turned out to be a very difficult problem. Robots are constructed of many components: computation, power supply, motors, sensors, structure, etc. Before the microprocessor revolution of the last 15 years computation in particular has been very large and heavy. Many well known efforts in the field of Artificial Intelligence [Nilson 84] [Moravec 82] involved large offboard computers as well as static environments. Despite using the most powerful equipment available these systems functioned extremely slowly and never operated in more general environments [Brooks 91]. These traditional robots were

built under the assumption that the best approach would be to build a complete internal model within the robot of the limited world, and then generate actions to navigate within this world. This proved to be very difficult for the restricted case, and impossible for the more general case.

Recently there has been a new approach to robot design [Brooks 85] [Brooks 91]. In this approach the environment itself is used to provide data on the world, and internal representations are kept to a minimum. Also integral to this approach is the concept of situatedness, that “the robots are situated in the world -- they do not deal with abstract descriptions, but with the here and now of the world directly influencing the behavior of the system” [Brooks 91]. At the MIT Mobile Robot (Mobot) Lab these techniques have been used to build a wide variety of robust robots. Some notables are Herbert [Brooks, Connell and Ning 88], Ghenghis [Angle 89], Attila [Ferrell 93], and Squirt [Flynn, Brooks, Wells, and Barrett 89].

In order for a mobile robot to be properly situated in the world it must be autonomous, that is, have its power and computation on board. Unfortunately much of current robot technology is pretty “clunky.” Battery technology in particular is very primitive. Modern batteries are heavy and are not sufficient to power state of the art computers and sensory systems. Large motors are needed to carry heavy batteries. In the end, if the newest most powerful components are used, the robot will be large, heavy, and slow.

By making the assumption that complete detailed models of the world are not necessary it is reasonable to base robots on the new generation of small microprocessors and microcontrollers that have arrived during the last 10-15 years. Processors such as the Motorola 6811 [Motorola] and the newer 68332 [Motorola 89] offer a reasonable amount of

processing power with very little power consumption. Mobot lab robots such as Ghenghis and Attila are fully autonomous. They have all of their sensors, motors, batteries, and computers on board. These small robots have used limited sensors such as contact sensitive switches, small bend sensors, and short range infrared detectors. While often quite useful, these sensors reveal only a tiny short range window on the world.

1.2 Robot Vision

In the design and construction of mobile robots vision has always been one of the most potentially useful sensory systems. For many animals vision is a very important sensory system. In primates (and therefore humans) vision processing is the primary sensory mode and occupies a large portion of the neo-cortex. While it is clear that a variety of senses are essential to a mobile entity, be it robot or animal, vision has a number of substantial advantages. It is able to provide a survey of a fairly broad section of the world (approximately 200 degrees in humans) and at some considerable distance. The regular interplay of light and surfaces in the world allow internal concepts such as color and texture to be extrapolated, making object recognition possible. Additionally, vision is a passive system, meaning that it does not require the emission of signals. No other mode of sensation has all these properties. Chemo-receptors (smell and taste) are inherently vague in directionality. Somatic (touch) input is by definition contacting the organism, and therefore provides no input from distant objects. The auditory system is probably the second most powerful mode, but in order to provide information about inanimate (and therefore silent) objects it must be an active emitter, like bat sonar. However, it is only underground and deep in the ocean that the environment is uninteresting in the visual range of the spectrum.

Despite this, a useful artificial vision system has turned out to be very difficult to implement. Perhaps this is because the complexity and the usefulness of the system are linked. Perhaps it is also because no other system must deal with such a large amount of input data. Vision receives what is essentially a three dimensional matrix as input, two spatial dimensions and one temporal dimension. This input's relationship to the world is that of a distorted projection, and it moves around rapidly and unpredictably in response to a large number of degrees of freedom. Eye position, head position, body position, movement of object in the environment just to name a few. In task oriented robot design it is desirable to use the vision system to reduce this huge amount of input and extract a small amount of reliable information which is useful for the task at hand. Whatever the reason for the difficulty in implementing useful artificial vision systems, it is clear from ourselves and other animals that vision is a useful and viable sense.

The Artificial Intelligence community has a tradition of implementing vision systems which are large, slow, expensive, and usually work only under fairly fixed conditions. Although there have been a number of larger vision-based robots using the traditional approach to robot design [Moravec 82] [Goto and Stenz 87] [Storjohann, Zeilke, Mallot, and Seelen 90], the size and computer requirements of vision have made it inappropriate for smaller mobile robots.

Recently at the MIT AI Lab Ian Horswill has built a \$20,000 robot named Polly which uses vision as its primary sensor system [Horswill 92a]. Polly is designed to give "brain damaged tours of the lab." It is capable of following corridors, avoiding obstacles, recognizing places, and detecting the presence of people. Polly's CPU is a 16 MIP DSP (Texas Instruments TMS320C30), and it employs a small video camera and a video frame buffer/grabber. These are all fairly small scale components by the standards of traditional

vision. However, it is the software and the philosophy that really sets Polly aside. It uses only a 64x48 pixel image, instead of the traditional 512x512, and simple algorithms that run in real time.

1.3 Philosophy of Low Cost Vision

It can be argued that intelligence is a kind of compression, and for vision this is particularly true. Vision brings in an array of data every frame, making a 3D matrix. At a standard resolution of 512x512 8 bit at 30 frames per second this is 7.6 Megabytes a second. However, it does not seem to be the case that humans extract more than 50-100 “facts” from vision every second. Even if 1K bytes are used to label a fact, this is around two orders of magnitude of compression. Human vision is at least an order of magnitude greater in resolution than 512x512, yielding roughly three orders of magnitude. Intrinsically, the greater the resolution one starts with, the greater the compression needed, and the more processing power required. In low cost vision, the goal should be to choose the minimum resolution that still contains the relevant data. There is an easy test for this: can we as humans see the needed information in an example picture.

When designing a vision-based robot that relies on the real world for its data storage it is important to ask oneself a simple question: “what basic facts do I want the robot to ‘see’ in the world around it?” Ideally one could then create a program that extracts this information from the image regardless of what environment the robot was in. However, this is not usually possible. One must always make some assumptions about the environment and the image. It is an illusion to think that the human vision system is without assumptions, we are just not usually aware of them. There are many basic assumptions that should hold true under almost any circumstances, for example, the relative relationship

between size and distance, or between position and location.

One should be aware of the assumptions one makes, and use them to reduce the computation required to extract the needed information. Ian Horswill has done significant work attempting to systemize the analysis of the process of simplifying vision algorithms based on assumptions [Horswill 92b] [Horswill 91].

1.4 Project Goals

For my thesis project I had a two part goal, first the design and construction of a small new vision system which would take the philosophy pioneered by Horswill's Polly system to a new level of miniaturization, and second the implementation of successful vision algorithms for use by a Martian rover.

The power and size constraints that apply to normal mobile robots apply even more so to interplanetary rovers. Mars is sufficiently far away that any rover must be largely autonomous. Every ounce of weight costs a fortune to get into orbit, and power is extremely scarce on an interplanetary mission. I set out to build a small self contained vision board that would contain frame grabbing, vision processing, and video display circuitry. The goal was to create a low power, low cost, small sized, realtime self contained system that could be added to robot systems without putting a big drain on their resources. This system could then use vision to determine a limited range of information and report this back to the host robot.

In the case of the Martian rover its job would be to quickly process the input in realtime and provide as output a small bandwidth of information reporting on the relative location of safe and unsafe areas for the robot to go. The system could also be programmed to guide the robot to simple targets such as its lander, or to large rocks for data collection.

1.5 Roadmap

Chapter 2 presents a short background in computer vision and a summary of some related work. Chapter 3 discusses the environment of the planet Mars and details the near term micro rover mission to that planet. Chapter 4 discusses the hardware system that I have designed, my design decisions, the tradeoffs I made, and the resulting performance. Chapter 5 provides a description of the algorithms used in low cost vision in general, and my system in particular. It also discusses the constraints of the Martian environment, and how they affected the design of my algorithms. In chapter 6 I discuss the results of my experiments with simple vision on the robot described in chapter 4 and provide some examples of test runs using the algorithms described in chapter 5. Chapter 7 is a summary of the work and a discussion of areas where further research would be useful.

Chapter 2

Related Work

2.1 Summary of Traditional Vision

Computer vision is broadly defined as the attempt to extract the same form of information from visual input that humans do. However, there is considerable argument about exactly what humans *do* perceive, and just as importantly what kind of internal representations humans hold in their heads. There are two major approaches to pattern perception: scene analysis and pattern recognition [Nevatia 82]. The first of these camps attempts to extract simple primitives from the input image, and then to recognize more elaborate scenes based on descriptions of these primitives. The second approach is based on the matching of the input pattern by more general algorithms to one of a set of known classes of patterns.

Traditionally an image has some “low level vision” processes applied to it, such as filtering and edge detection. After this the image is frequently analyzed to construct what is called a “sketch,” a list of the lines in the image. Vision algorithms will often attempt to isolate objects from this sketch by matching components against templates. Most of these processes are very processor intensive, particularly sketch generation and object recognition. Traditional vision can be viewed as the complement of three dimensional rendering. In rendering a detailed three-dimensional model of objects, lights, and surfaces is used to generate a two dimensional image. In traditional vision, the 2D image is analyzed

in the attempt to generate this same detailed 3D model of the world. 3D rendering is a very successful computer technique, however its complement has been much more difficult to implement.

Perhaps this is because there is less information available in the 2D image, and vision has to reconstruct it, making assumptions about the nature of the world in order to do so. Perhaps the encoding of information in the 2D image is more complex. In actuality the visual input to an active agent is not even a simple 2D image, but one that changes over time and in relation to the actions of the agent.

The traditional approaches usually operate at resolutions at 512x512 or greater, and require seconds at best to complete their processes. Often they take minutes or hours per frame to compute. It is clear that this is orders of magnitude too slow for use in robots that can act on the same time scales as human and animal agents. In addition it is not clear that humans have a complete 3D model of the world in their heads, in fact I would argue that this is not the case.

2.2 Low Computation Vision in Insects

Insects as a general rule have vastly smaller nervous systems than vertebrates, particularly mammals. Nevertheless, they often make extensive use of vision as a sensory system, and at a level of competence far exceeding most of our efforts to duplicate. In this section I am going to show some examples of research that indicates that insect vision is often quite simple in its computational demands, taking constancies of the environment into account to simplify the calculations.

For example, it has been shown that ants and bees can derive compass information “not only from the sun [Santschi 11], but also from small patches of blue sky [Santschi 23], and

that in the latter case they relied on the perception of polarized skylight [von Frisch 49]” [Wehner 89]. The intensity and color characteristics of the sky are highly variable, and the sun is often obscured. There are two components of the sky’s polarization state, direction and degree. The first of these, the e-vector, is fairly immune to weak atmospheric disturbances like haze, pollution, and clouds. These insects have a specialized part of the retina (POL area) which is sensitive to the pattern of polarization found in the sky. When this area is lined up with the solar meridian (the arc traveling from the horizon through the sun to the opposing point on the other horizon) it fires aggressively. In order to find the direction of solar travel, the bee need only rotate until this area of its retina is maximally stimulated [Wehner 89].

In this example the insects have exploited a constant feature of the environment, that the polarization pattern is reasonable immune to atmospheric variation. Polarization is inherently sensitive to rotation, and is therefore ideal for determining orientation relative to the source of the pattern, in this case the sun. The insect has evolved a hard wired system which is tailored to this specific reality of its world. The system is quite specialized, but it uses the structure of the world to make possible an extremely low computation solution to a difficult vision problem. This method works precisely because it is not general, but because the system uses the constraints of the environment to extract an extremely small amount of information from the large input. In small nervous systems like that of insects it is not possible for them to deal internally with enormously complex data, they are forced to reduce their world to simple terms.

Honeybee navigation presents an even more interesting example. While most robot navigation systems employing vision have been based on stereo, honeybees have their eyes mounted on the sides of their heads in such a manner as to give them almost no stereo field.

Instead it has been shown that bees use optical flow measurements to guide their flights [Srinivasan, Lehrer, Kirchner, and Zhang 91] [Srinivasan 92]. Optical flow is a measure of the speed and direction of movement in the image, and for a moving robot the distance which an object moves in the image is proportional to the inverse of its distance (from simple geometry). This means that closer objects will “flow” much faster than farther objects. The shape of the $1/d$ curve is such that nearer objects move much faster than farther objects. By turning into the direction of least flow, a moving agent will be turning toward the direction with the most space. It will turn away from looming obstacles.

Recently there has been some interesting work applying these techniques to robotic research [Sandini et al 93] [Coombs and Roberts 92]. Sandini et al have designed a robot which has two laterally mounted cameras. The magnitude and direction of the flow as seen by each camera is calculated, and the robot attempts to balance the flow on either side of it by turning into the direction of least flow. This approach does not build any extensive internal representation of the world, but instead relies on realtime sampling of the real world to give it up to date information. Additionally, it does not require any calibration because all “distances” are relative measurements rather than absolute calculations of distance.

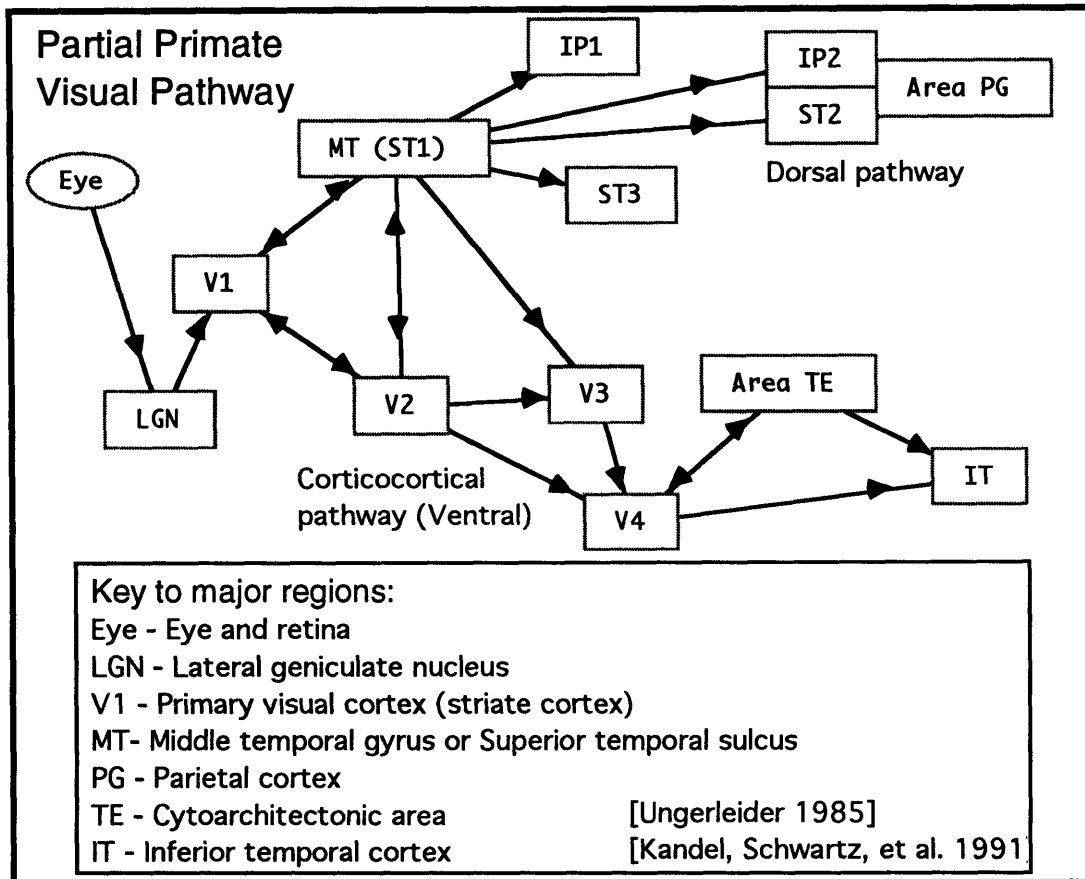
2.3 Summary of Primate Vision

The human visual system is often causally regarded not only as the target of vision research but as a fully general system. This is clearly not true. While our visual performance is extremely impressive, and far beyond anything we are able to build with our current level of technology, it is not completely general. For example, people are usually unable to recognize the faces of familiar people if their images are viewed upside-down.

Below I summarize some of the macro-features of the primate visual pathways. The primate visual system is a kind of branching pipeline, where information flows forward from the retina to layer after layer of progressively higher level processing areas. However, the street is two way, each layer projects as many neurons back to previous layers as it does forward.

In higher vertebrates visual information enters at the eye and undergoes processing in parallel at the retinal level. The information then flows via the optic nerve to the Lateral geniculate nucleus (LGN) and from there to the primary visual cortex (V1 or striate cortex) at the back of the head [Kandel, Schwartz, et al. 91]. A substantial amount of primate cortex is devoted to visual processing. There is extensive evidence from "anatomical, electrophysiological, behavioral, and metabolic mapping studies that visual cortex in the monkey" is a pathway that begins at the primary visual cortex, or striate cortex, and includes prestriate cortex and parts of the temporal and parietal lobes" [Ungerleider 85]. These areas represent a system for visual processing that is both structurally and functionally heterogeneous.

Present knowledge suggests a divergence of information flow coming out of striate cortex (V1) into two separate pathways. "One pathway is directed dorsally into posterior parietal cortex [PG] and the other is directed ventrally into inferior temporal cortex [IT]" [Ungerleider 85]. It is believed that these two pathways represent separate pathways for spatial perception and object recognition, with PG representing the former and IT the latter [Ungerleider 85]. Studies examining the effects of lesions on behavioral tasks have shown each of these areas to be essential for the performance of tasks in their respective skills of spatial perception and object recognition while having little or no effect on the performance of the other group of tasks [Mishkin, Ungerleider, & Macko 83].



"The corticocortical pathway from striate cortex into the temporal lobe plays a crucial role in the visual recognition of objects" [Desimone, Schein et al. 85]. Anatomic studies point to a pathway that is organized as a serial hierarchy of multiple visual areas, including V1, V2, V3, V4, and inferior temporal cortex (IT). This hierarchy appears to process multiple attributes of the visual stimuli in parallel at increasingly higher levels of abstraction. At each level in the hierarchy the receptive field of a given neuron grows, increasing the area of the visual field that the neuron is responsive to. Between area V4 and IT, retino-topic organization is lost. Neurons in the IT have receptive fields that cover most of the visual field, oriented largely in the center. This indicates that they are sensitive to characteristics

of the visual field independent of position. Areas such as V4 and the IT are sensitive to many kinds of information relevant to object recognition. For example, many V4 cells are sensitive to spatial qualities (e.g., direction of motion, orientation, length, and width) as well as spectral qualities like wavelength. This illustrates how V4 is not responsible for the processing of merely one attribute of a visual stimulus, but several in parallel. Neurons in V4 have small excitatory receptive fields and large suppressive surrounds. They appear to be involved in recognition of object components that are then combined into more complex forms in the IT. The evidence indicates that in the occipito-temporal pathway, many stimulus qualities are processed in parallel, but the form of the processing becomes more global at each level [Desimone, Schein et al. 85]. Neurons in the later neural area appear to process large parts of the visual field, often including the center. They do so for multiple attributes as in the V4 area. "Many IT neurons, for example, are selective for the overall shape, color, or texture of a stimulus, anywhere within the central visual field" [Desimone, Schein et al. 85].

2.4 Previous Simple Vision Work at MIT

My work here at MIT owes a great deal both practically and philosophically to some previous low computation vision projects at the AI Lab. There has been some earlier work on low computation solutions to the problem of visual navigation. One project [Brooks, Flynn, and Marill 87] involved the use of self calibrating stereo vision. This system simplified the vision task by examining only one scan-line from each camera, but by tracking them over time. The small computation requirements of the approach allow a high frame rate which simplifies the problem of tracking features in the image. Strong vertical edges in the stereo map are tracked over time. In a forward moving robot, lines produced

by oncoming objects will move outward from the center of expansion of the camera. The location of this point in the image is easily accomplished. From this information it is possible to calculate the time to collision for each of the tracked features. This system represents stereo vision that does not require extensive calibration or foreknowledge of the camera parameters and geometry. Human vision is able to adapt to different calibration requirements extremely rapidly. The reliance of numerous specific calibrations and conditions is a major weakness of many artificial vision systems.

The work that most closely influenced my own is Ian Horswill's work [Horswill 93] in constructing a reasonably cheap yet robust vision based robot [Horswill 92a] and in systemizing the process of simplifying vision algorithms [Horswill 92b]. Ian has built a robot named Polly which navigates in an unmodified office environment containing both static and mobile obstacles, as well as dynamic illumination and layout. Except for some minor feedback from its motor base Polly has only one sensor: vision. It is able to use this broad band sensor to avoid obstacles, detect locations, follow corridors, and detect people. Polly cost only about \$20,000 in 1991-92 parts and has been run successfully for hundreds of hours.

The hardware on Polly is quite straightforward, although extremely minimal by the standards of most vision robots. It contains a 16 MIP DSP CPU, a frame grabber, minimal I/O circuits, and a small wheeled base. The robot is fully autonomous, carrying all its computation and power onboard for a reasonable length of time (a few hours). It is the software however that is unique.

Polly views its world at the resolution of 64x48 8 bit gray, allowing it to process 15 frames per second. It is "heavily I/O bound however, and so it spends much of its time

waiting for the serial port” and bus [Horswill 92a]. Polly makes a number of assumptions in its program that allows it to extract useful information from its images quite quickly.

The robot has an obstacle avoidance routine where it makes some of the following assumptions:

- The carpeted floor (there are several kinds of carpet) has no low frequency texture (background texture constraint).
- The floor is flat (ground plane constraint).
- Obstacles sit on the floor, and usually have some texture.

For robots, avoiding obstacles usually consists of calculating a map of freespace around the robot. This is essentially an estimate of how far in every direction there is open floor until an obstacle. Because of its assumptions Polly can separate which pixels should be considered floor and which should be considered obstacle. First it blurs the image, effecting a low pass filter to remove noisy areas from the floor. Second, it detects edges on this image. Because it has assumed that floors have no texture, areas with edges must be obstacle. Thirdly, because it knows the floor is flat and that obstacles sit on the floor, it can assume that higher in the image plane is farther, and that anything above texture is behind an obstacle. When it scans up each column in the image until texture is encountered this becomes the measure of free space in that column. Polly is uncalibrated, and only measures the relative distance of obstacles, and the relative horizontal location. Because it runs in realtime it can merely decide which side, left or right, contains the most free space until an obstacle is encountered, and turn in that direction. The realtime nature of the process will compensate for aberrations. The system only requires that it can decide which way would be best to turn “just a little, right now.” Polly moves at around a meter per second, and is an able avoider of obstacles. A few circumstances such as transparent glass or drop-offs violate its assumptions and give it trouble, but by and large it moves around the cluttered

lab without mishap.

Polly also has a corridor follower of similar design. Knowing that office corridors have many lines extending parallel to them, by calculating the convergence of all lines in the image it is possible to estimate the vanishing point of these lines, and hence the axis of the corridor. It uses the previously mentioned freespace map to find the distances to the nearest objects on the left and right, presumed to be the sides of the corridor. The robot then attempts to maintain an equal distance between walls, staying in the center of the corridor.

In addition to these lower level systems Polly uses low resolution templates (16x12) to identify locations. The high level systems in the robot are designed to give "brain damaged tours of the lab." It drives around the floor until it detects a person, and then it offers to give the tour. If they accept by shaking their leg it will lead them around once, announcing various locations. The robot has been seen to do this on its own for at least an hour or two without mishap.

Polly illustrates that high resolution and computation extensive algorithms are not necessary to do useful vision work. It compensates with intelligent analysis of the makeup of its world and realtime processing for quick response.

Chapter 3

Mars

Autonomous mobile robotics have the potential to be of great use in the area of interplanetary exploration. This distances involved make it extremely difficult to send manned missions to other planets in our solar system. Manned craft are heavier and larger than machine only vehicles, costing more to place in orbit. In addition, the travel time to even Mars and Venus, our nearest neighboring planets, is on the order of a year. The air, food, and water required to support a human crew for both the outward and return journey make the cost prohibitive. Robots however can sleep during the long voyage, consuming only a small amount of power. They can also be designed to survive the extreme temperature ranges mandated by space. On the moon, with its light delay of less than a second, tele-operated robots are feasible. However, on Mars, with a light delay of tens of minutes this possibility is extremely limited. In addition, there are severe constraints on radio telescope time so that constant communication is very difficult to maintain. Overall, autonomous and nearly autonomous robots offer the best potential for near term exploration of the Martian surface.

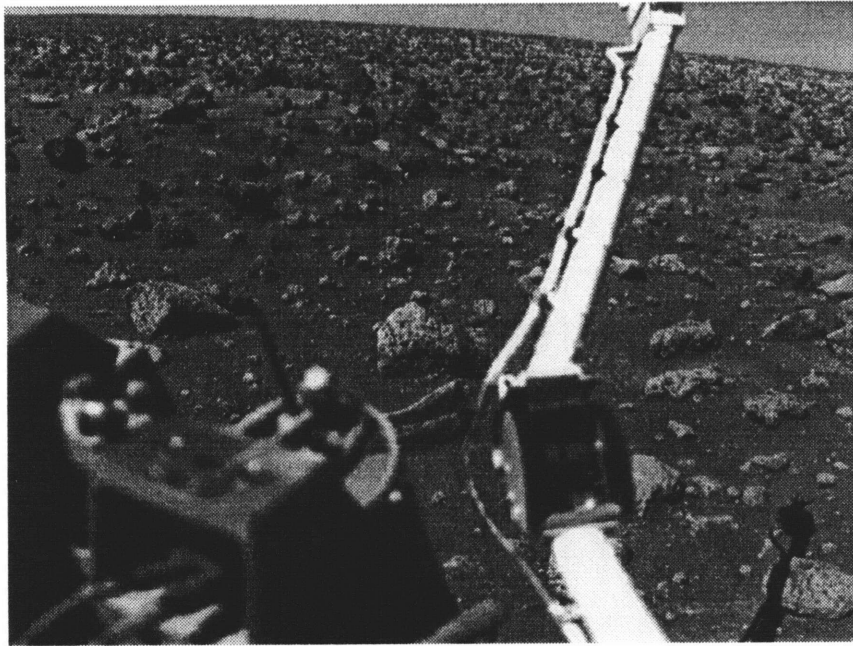
The characteristics of the system I have built are ideal for the limitations of interplanetary travel. Power is critically limited, as is size and weight. Low power means low computation, because for a given silicon technology, power consumption is directly proportional to the amount of processing power. My vision boards, or other similar

hardware, when equipped with intelligently designed low cost vision algorithms can provide extremely useful sensory information to an interplanetary rover.

3.1 Features of the Martian Environment

Mars is the fourth planet in our solar system. It is possessed of an atmospheric pressure 1% that of earth's and is much colder than our planet. Consequently it has no liquid water, although it does have water-ice clouds. The thin atmosphere results in a large daily temperature swing, up to 190K to 240K during the summer. The surface is rocky with large quantities of thin dust. There are local dust storms every year, and sometimes global dust storms that cover most of the planet. Most of the time surface winds are less than 15 meters per second. The atmosphere is predominately carbon dioxide, with some free nitrogen. The Martian surface receives around 66% of the sunlight that earth does. This amounts to an average solar radiation of 500 watts per square meter, about the same light received during a bright day in a major earth city [Nasa 88].

The surface of Mars as seen from the two Viking landing spots in 1976 is a flat and barren place (see adjacent picture). "Dune like drifts are superposed on a rock substrate and blocky rims of near and distant large impact craters" compose the basic landscape [Nasa 88]. "Rocks of a variety of sizes, shapes, and morphologies are embedded in or resting on the substrate; large rocks locally protrude through the superposed drifts. Sizes of nearby rocks range from a centimeter or so to several meters. Surfaces of the substrate between the rocks are covered with a thin veneer of fines or littered with small mm to cm size objects which are chiefly clods." There may be local accumulations of bright red dust covering the surfaces of both Viking sites [Nasa 88].



Mars from the Viking Lander

The surface materials have been classed into four categories: (1) drift, (2) crusty to cloddy, (3) blocky, and (4) rock. Drift material is believed to be a reddish dust similar in consistency to baking flour. Drifts of this material could be a potential hazard to a robot if the device was to sink below its wheel base, or if the traction was sufficiently poor. Crusty to cloddy material is believed to be more cohesive, composed of larger firmer clods. Like the drift it is usually reddish in color. Blocky material is much more cohesive than other two, forming large clumps. It also tends to be reddish, but is occasionally darker. Rocks vary in density from very dense and solid to more porous. However all rocks within reach of the Viking landers were solid enough so that the lander shovel could not scratch them. They tend to be a dark gray in color. The rocks are scattered in a even distribution of sizes.

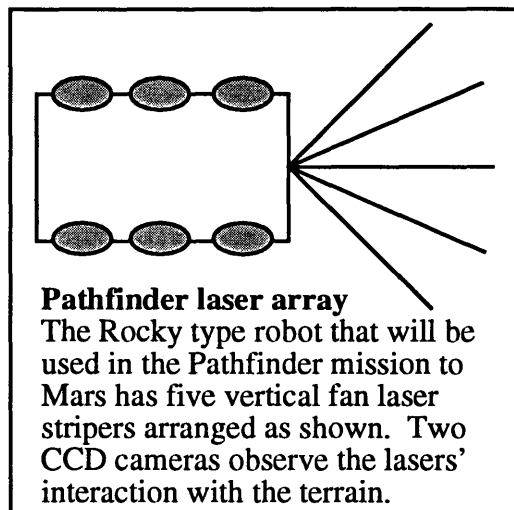
3.2 The Mesur/Pathfinder Mission

Nasa has scheduled an unmanned mission to Mars for launch in November of 1996. This mission, called the Mesur Pathfinder Mission, is to include a microrover with a mass under 10 kilograms that will be designed to traverse over the terrain within a few tens or hundreds of meters from the mission lander. It will explore goal points targeted from the ground and selected from the lander's stereo cameras [Wilcox 94] [Weisbin 93]. The lander is equipped with two 256x256 pixel CCD cameras. The rover and lander stay in communication via a modem [JPL 93].

The pathfinder rover, a Rocky series rover [Wilcox 94], uses a six wheeled "rocker-bogie" mechanism. The rover has angle and position sensors on its "rocker-bogie" mechanism and shaft encoders on its wheels. It is equipped with three CCD cameras and five laser stripers. It uses the cameras and the lasers together to determine the location of geometric hazards and combines these with wheel information to help locate non-geometric hazards.

Because of the power requirements imposed by interplanetary travel the rover uses a digital camera system similar to the one I have designed. It clocks the signal from the CCD directly into the processor. In order to reduce the computational requirements, it uses the laser range finders to enable it to examine only a few lines in the image. The rover fires two of its lasers in a vertical fan

each frame, then processes three different scan lines in order to get estimates for three



different distances. A laser filter on the CCD allows the rover to find the horizontal location of the laser stripe in the given scan line. From this position an estimate of the height of the terrain in the direction of that laser and at the selected distance [JPL 93] is fairly easy.

This method is another good example of a low computation vision system. The laser strippers are used as a tool to reduce the amount of the image that needs to be processed. There is a cost however, for the lasers consume significant power, their range is limited, and it is only possible to find obstacles that intersect the laser paths.

In addition to the use of the CCD laser system for the detection of geometric hazards the rover is programmed to deal with the danger of sinkage and slippage in the Martian drift material. The visual system provides an estimate of the visual height of the soil in front of the rover. When it reaches this area it can use the relative position of its front wheels in comparison to their expected position to estimate how far into the drift, if any, the wheels have sunk [Wilcox 94].

3.3 Constraints of the Martian Environment

The locations visited by the Viking landers were selected for their flat terrain, and as observed in the photos this is so. The basic contour of the land consists of flat plains, with an occasional gently rolling hill or dune. If one ignores the rocks then the *ground plane constraint* holds to a certain extent, especially in the local environment. However, the rock terrain, and the uncertain quality of the ground means that there is no guarantee that the rover will remain exactly perpendicular to the ground.

Visually the world consists of a uniform sky, probably brightly illuminated, and a darker ground plane. The location of the sun, or at least the brightest area of the horizon

(indicative of the solar meridian) should usually be easy to distinguish. The angle of the bright sky against the darker horizon should be easy to determine. The terrain consists of two basic features, rocks and dirt/dust. The rover will be capable of driving over rocks of a certain size (probably about half its wheel diameter), and no larger. Presumably it can drive over any solid dirt, but there might be dust pits that it will not be able to navigate. Unfortunately, dust and dirt look pretty much the same.

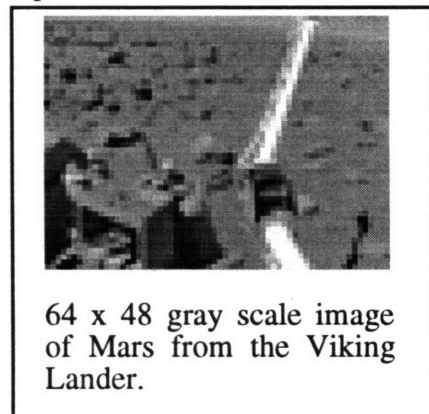
The rocks however have different visual characteristics than the dirt/dust. The latter is light color, and relatively free of low frequency texture. The rocks are usually darker, or sometimes bright with dark contrasting shadows. Their edges, and the separation between their specular highlights and their shadows are quite textured. In addition to these intensity features the rocks protrude from the ground, giving them different stereo and motion characteristics than the dirt/dust.

Chapter 4

The Mobot Vision System

4.1 Image Size, Frame Rate, and Processor Speed

In accordance with our philosophy I decided on an image size of 64 by 48 with 256 levels of gray. These images are large enough that most important details of the view are clearly visible and recognizable to humans (see adjacent example image). Any number of gray shades in excess of about 64 are nearly indistinguishable, but 256 is convenient and allows for a neat single byte per pixel representation. My chosen resolution requires an image size of 3K bytes. I also chose a frame rate of 10-30 frames per second. This level of frame rate, which is often referred to as a realtime frame rate, is crucial to the kind of system I was building. The active agents that we are familiar with: people and animals, all live in a time scale where they take a small number of “informed” actions per second. Many traditional vision systems run at frame rates of once every few seconds, every few minutes, or even every few hours. This is not fast enough to make any decisions at a rate which is similar to that of the intelligent agents we are used to. Traditional vision systems also build a complex internal map of the world from which they can draw information between frames. I however believe that this is an overly complex and counter productive strategy, but consequently



need to compensate by sampling the real world at a much higher rate. It should be noted however, that on interplanetary rovers power and weight constraints require a much slower rate of travel as well as much smaller computer hardware. These two reductions are roughly in scale so that while my algorithms require 10-20 frames per second for good response time at one meter per second, at a one centimeter per second rate of travel a few seconds of processing per frame is sufficiently fast.

Since my goal was to provide vision for a robot moving at approximately a meter a second and capable of making decisions at a similar rate to that of biological agents it was necessary to target a frame rate of at least ten frames per second, preferably thirty. Combined with the chosen image size, this yields a data rate of between 30 and 90 kilobytes per second.

In building any computer system the choice of processor is of vital importance. A useful strategy would be to select the processor with the lowest power consumption and cost that was capable of handling the required processing. I chose the Motorola 68332. This is an integrated 32 bit microcontroller with on board RAM, serial hardware, time processing unit (TPU), and a peripheral interface unit. It is capable of executing approximately two million 32 bit instructions per second. This means 64 instructions per pixel at 10 hertz or 22 at 30 hertz. These are reasonable enough numbers to run the kind of algorithms needed for low cost vision.

The 68332 was available in a simple one board solution from Vesta Technologies. To this we added 1 Megabyte each of RAM and of EPROM.

4.2 Grabbing Frames: Digital Cameras

Electronic Cameras are all based on a kind of chip called a Charge Coupled Device, or CCD. These chips have a grid of “charge buckets,” which are sensitive to photons. When light strikes them, there is a measurable electron build up. CCDs typically have a number of standard phases and controls. They have a blanking phase, which clears the charge buckets, an integration phase, which is equivalent to a camera’s exposure time, and a read-out phase. Most CCDs have an Integration signal (INT) which turns on the integration phase, and an image signal which clocks all of the pixels on the CCD down one line, and a shift signal, which clocks one pixel out of the bottom most line. In this manner CCD’s are read serially during their readout phase, clocking one pixel at a time out of each line, and then clocking each line in turn.

A traditional video camera consists of a lens with a CCD behind it. These video CCDs generally have a resolution of around 768x512, similar to that of the NTSC video signal. The camera will have a number of driver chips used to produce the bizarre voltages that CCD’s require. In addition they will have a set of timing chips to produce the CCD timing signals, and an armada of converter chips to transform the output of the CCD into a 60 hertz analog video signal. A traditional frame grabbed will then decode this complex signal, and digitize it at approximately 640x480, either at 30 or 60 hertz. This whole process of video encoding and decoding too costly in terms of time, chips, and power for a low resolution digital camera system intended for robot use.

While the use of NTSC video does allow the convenient use of off the shelf cameras and monitors, an electronically much simpler approach is to connect the CPU directly to a low resolution CCD. With the help of visiting scientist Masaki Yamamoto I built a small piggy back Camera Board containing all of the hardware needed to allow a small processor

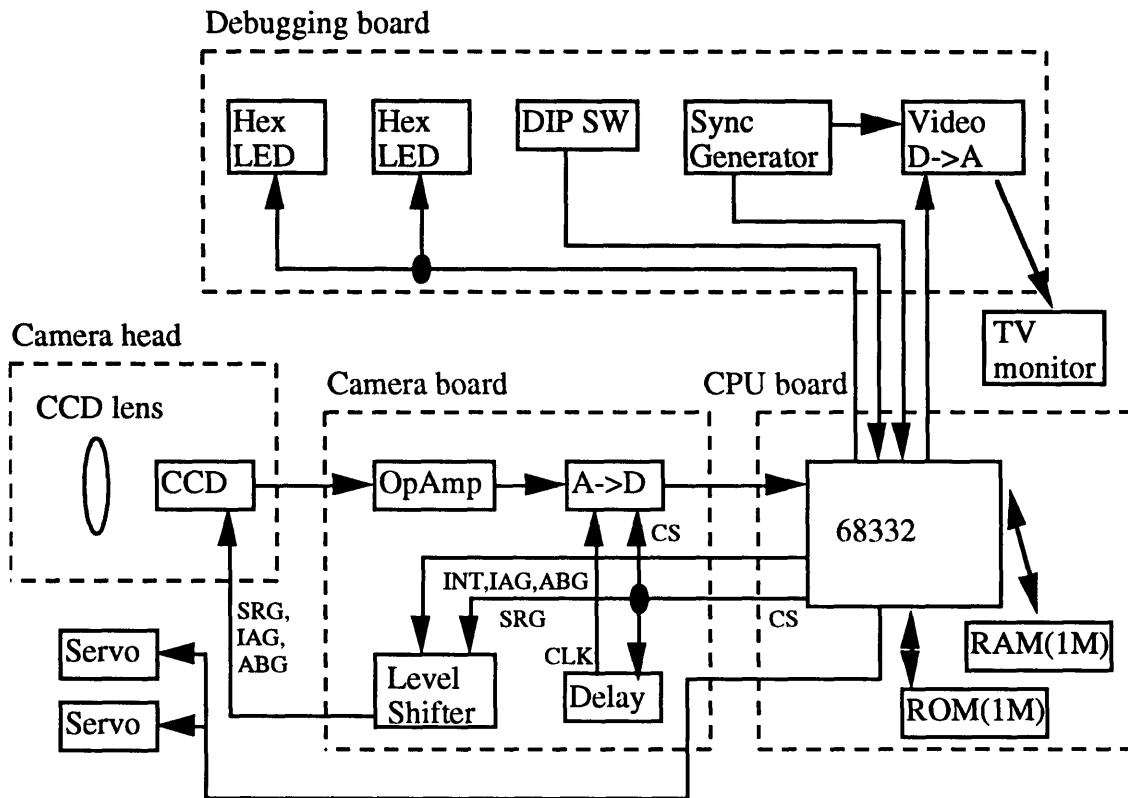
to grab low resolution frames digitally.

4.3 The Camera Board

The Camera Board is the heart of the system. I chose a low resolution CCD from Texas Instruments (TC211) [TI92]. This chip has a resolution of 192 by 165, closer to the 64x48 that I wanted, and in the same aspect ratio. The chip is also simple to drive, it has an integration signal (INT), an anti-blooming signal (ABG) used to reduce the bleeding of electrons from one charge bucket to another, an area signal (IAG) which clocks rows down the CCD, and a serial shift signal (SRG) which outputs pixels. The CCD requires +12 and -12 voltage while the rest of the chips in the system are all TTL (+5 V and ground). We used an Analog Devices converter to provide the voltages for the CCD.

Also on the Camera Board is the level shifter circuitry used to drive the CCD chip. This was specially designed with both simplicity and low power consumption in mind. The CCD chip requires its clock signals to be at specific analog voltages and so we explored three methods of converting the processor's TTL level timing signals. The obvious method was to employ the driver chips sold by the CCD manufacturer for this purpose. We rejected this because of the high power consumption which seems to be unavoidable in this kind of high speed clock generation circuitry. The second option was to use an operational amplifier to add analog voltages. Because we wanted a low power circuit, and also wanted to reduce the number of components, we chose a third solution, which was to use analog switches that could toggle the voltage at a reasonably high frequency and which were fast enough for the processor's clock rate. Our circuit resulted in a total power consumption for the Camera Board of less than half a watt (when it is supplied +5V, +12V, and -12V directly).

The 68332 has an onboard TPU which is capable of running simple timer programs



Block diagram for the Robot Vision System

on 16 I/O lines with very little processor intervention. This device was easily programmed to drive the ABG signal, while the INT and IAG signals were connected to simple parallel pins on the processor. The SRG signal was the most difficult signal to produce because it is the fastest and most timing sensitive. The SRG clocks out individual pixels, and during the CCD's read-out phase it needs to occur two million times a second (2 megahertz) and needs to be synchronized to the pixel digitization. However, there is a clever solution to the problem of producing this signal. We connected the output of the CCD to a small amplifier and then to a high speed Analog to Digital Converter (Sony CXD1175). This last chip was

mapped as a simple memory device to the CPU. When the CPU attempts to read this memory device it automatically produces a chip enable signal for each read. By running this signal through a short delay it becomes appropriate to drive the SRG on the CCD. The Camera Board also has several adjustable potentiometers, an adjustable delay knob, a signal offset knob, and a signal gain knob. All must be adjusted together in order to achieve a good picture.

4.4 The Camera Head

On the Camera Board is a socket where the CCD can be mounted directly, or it can be connected via a six wire cable to a camera head. Since the robot needed to insure as wide an angle as possible, we explored small short-focal-length lenses. Generally wide angle lenses have several merits, such as a wider depth of focus, which makes a focusing mechanism unnecessary, a smaller F number, which increases the image brightness, and an insensitivity to camera head vibration. However, it is sometimes difficult for wide angle lenses to compensate for image aberrations. After testing several aspheric lenses, pinhole lenses, and CCD lens sets, we decided to use a $f=3\text{mm}$ CCD lens from Chinon (0.6" in diameter and 0.65" long, 5g weight).

In front of the lens we placed ND filters in order to prevent over saturation of the CCD. Our CCD is actually quite sensitive and needs at least a 10% pass filter to operate in room level lighting, sometimes it even needs considerably more. In order to expand the dynamic range of the camera the frame grabbing software is designed to calculate the light level of the image and adjust the integration (exposure) time of the frame correspondingly. This adds an extra 10 decibels of dynamic range to the system, allowing it to work adequately in subjective indoor light levels ranging from lights off at night to sunlight

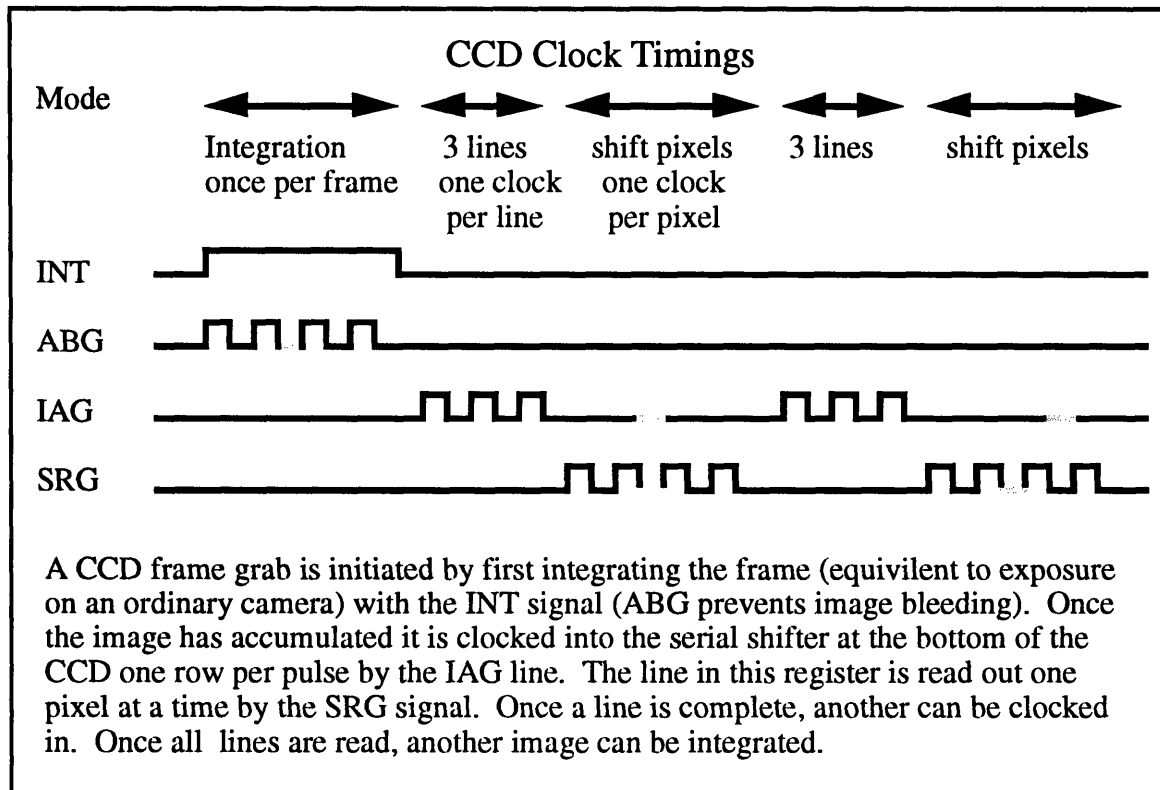
streaming in through several large windows.

The camera is mounted on top of two Futuba model airplane servo actuators (FP-S5102). These are very small and light weight, and allow easy and fairly precise control of the camera position by the CPU. The servo actuators are connected via the Camera Board to the 68332's TPU. This allows the generation of their Pulse Width Modulated (PWM) control signals with virtually no CPU overhead. These actuators give the camera both pan and tilt over a fairly wide range (170 degree pan, 90 degree tilt). The CPU has a number of simple routines that allow it to specify both absolute and relative positioning of the actuators, to read where they are, and performs bounds checking to prevent the actuators from twisting the camera into themselves. The camera head and its servo actuators weigh 68 grams.

4.5 The Frame Grabber Software

The circuitry on the Camera Board provides the CPU with a serially accessible 192 by 165 image memory. The exact resolution digitized is a software option. I typically sample at my chosen resolution of 64x48 by taking every third pixel, a process known as sub-sampling. The focus on the lens can be used to adjust the sharpness. The software program required to grab a frame is very simple. I merely clear the chip by clocking all the rows off, then integrate by toggling a parallel bit. During the readout phase I clock a line, and then read 165 bytes from the input address, every third of which I store. Due to the time intensive nature of this process I unraveled the loop. It takes approximately 7 milliseconds for the processor to clock and read in a single frame. At 10 frames per second this only consumes 7% of a 16MHz 68332's processor time. This allows 93% for processing the images, which amounts to about 186,000 instructions per image, or about 62

instructions per pixel. 62 instructions per pixel, while it might not seem like much, or more than sufficient to do many of the simple vision tasks we have in mind. If more calculation is required, then a faster version of the chip can be used to double the available processing power.



4.6 The Debugging Board

I also designed a second piggy back board which attaches to the CPU and Camera Boards. This Debugging Board is optional, and not required for the system to grab and process frames. The Debugging Board has 2 hex LEDs for output, 6 bit DIP switch for input, a reset and debug button, and some video circuitry. These circuits are extremely simple because the 68332's interface module allows nearly any kind of device to be mapped

onto the bus. Even the video circuitry is very simple.

In doing any kind of vision based work it is essential to be able to see what the camera is looking at, and extremely useful to be able to view internal images produced by the programs. However, because our camera is completely digital, it is not possible to simply connect a video monitor to the system. I evaluated a number of video output options, and rejected the conventional approach of building a frame buffer and video output system because it would be too complex and require too much power. Since the software based approach had worked so well for grabbing frames I decided to use it for output as well.

We used two chips, a video level Digital to Analog Converter (Intech VDAC1800) and a video sync generator. The timing signals are run into the extra two input bits in our 8 bit input port and a trivial program on the CPU watches for transitions and outputs image data to the D to A convertor at the appropriate times. This allows me to display images contained in the CPU at 30 hertz to a video monitor with just two chips. Most of the work is done by the CPU. The system is capable of capturing images from the camera and outputting them to the video display at 30 hertz. Since video output consumes considerable CPU time it is not advisable to run the display at the same time as doing extensive vision calculations. However it provides a nice, easy output of the camera image in realtime which is essential for debugging, and for tuning the camera circuitry. There are also two switches on the Debugging Board to toggle the LED and video hardware for power consumption reasons.

4.7 Cost Analysis of the Hardware System

In the design of the Mobot Vision System, Masaki Yamamoto and I went to a great deal of effort to minimize the power consumption and were quite successful. The CPU board consumes 0.5 watts of power. The Camera Board also uses 0.5 watts. This means that the entire CPU and vision system consumes less than 1 watt of power. The video out circuitry on the Debugging Board requires an additional 1 watt of power, however there is a switch to disable this circuit, and since its use is for debugging this is not significant. The servo actuators also require some power. When idle they consume a mere 0.05 watts. Unless they are being moved constantly their average power draw is barely more than this.

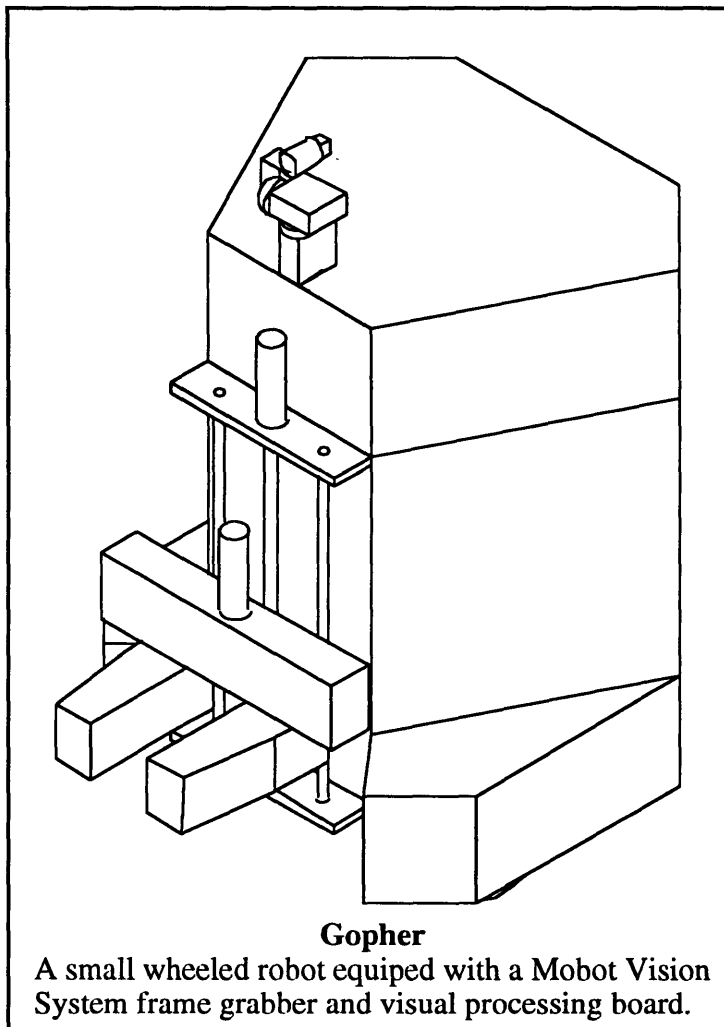
Cost was also a significant factor in our design. In 1993 dollars the Vesta CPU Board costs \$300. One megabyte of static RAM costs \$200 (one could use less to save money, a megabyte of EPROM costs \$25 (again one could use less to save money), two servos cost about \$130, the CCD costs \$35, the driver chips \$30, the analog to digital convertor \$25, the video chips \$50, the power convertor \$65, and miscellaneous other chips about \$10. This is a total cost of around \$700, many significant components of which could be eliminated to save money. One could use less RAM, or forego the servos or Debugging Board, possibly bringing a useful system as low as \$350 in parts.

Overall this system is a small, cheap, and low power vision solution. It provides the input of 64 by 48 pixel 256 level gray scale frames at 10-30 hertz from a small camera with CPU controlled pan, tilt, and dynamic range, as well as about 62 680x0 instructions per pixel of processing time at 10 frames per second. All of the electronic circuits fit in a space 15 cubic inches big, consume less than a watt of power, and cost about \$700 dollars to build. The available processing time is sufficient to do simple calculations such as blurs, edge detections, subtractions, the optical flow normal, thresholding and simple

segmentation. A number of these can be combined to provide useful realtime vision based information to a robot.

4.8 Gopher: A Vision Based Mobile Robot

In order to fully test our system in a real environment we also built a small vision based robot, Gopher (see diagram). This robot is based on a R2E robot from ISR. The robot is quite small (15 inches tall, 10 inches wide, and 12 inches long). It is wheeled, has a gripper on the front, and a number of sensors, including shaft encoders, touch sensors, and infrared sensors. These motors and sensors are controlled by a number of networked 6811 processors. A master 68332 processor, runs the behavior language which was designed here at the MIT Mobot lab [Brooks 90]. We have added a flux gate compass accurate to 1 degree and two of our 68332 based vision systems. The boards are all contained within the R2 case, which has been extended vertically for this purpose.

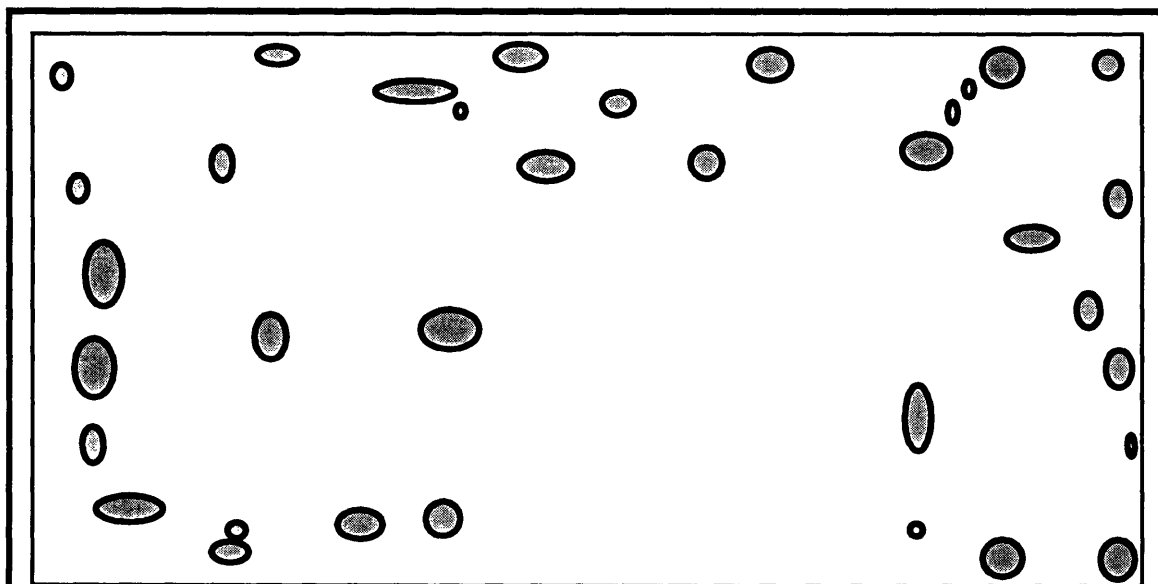


pose. We have mounted one of our CCD camera heads on its dual servo base on top of the robot, adding an extra 3 inches to the robot's height.

The Gopher robot provides a small and relatively simple low cost robot with an independently controlled camera and vision sub-system. I have used this system to implement many simple visual based tasks in a coordinated and integrated fashion.

4.9 Sim Mars: A Simple Artificial Environment

In order to test the Gopher robot in a Mars like environment I have built an artificial Mars surface area on the ninth floor of the MIT Artificial Intelligence Laboratory. This area consists of a pen, some twenty by thirty feet which has been covered with a sandy yellowish flooring and low yellow walls. Scattered throughout the area in a random pattern are



Sim Mars

This illustration represents an overview of the "Simulated Mars" environment. The Area is 30 by 18 feet. The gray ovals mark the location and approximate sizes of the rocks in the area. The sun, a 750 watt halogen lamp, is located in the lower right hand corner.

several hundred pounds of gray volcanic rocks. I have also purchased a high intensity light to simulate the bright Martian sunlight.

The roughly uniform floor (which actually has numerous stains and dimples) is supposed to simulate the dust and dirt of the Martian surface. The volcanic rocks, ranging in size from two to eighteen inches in diameter are fairly similar to the rocks in the Viking images. My ground is however much flatter and smoother because the Gopher robot has very small wheels and is incapable of transversing anything but a smooth surface. The density of small rocks on Mars is also much larger for this same reason. Mars has an average illumination of 500 watts per square meter. To light my simulated Mars area to this degree of brightness would take many kilowatts of lighting. This was too expensive and hot to implement inside, but the area is still lit by a single bright source.

Chapter 5

Software

5.1 Images: How good is 64 by 48

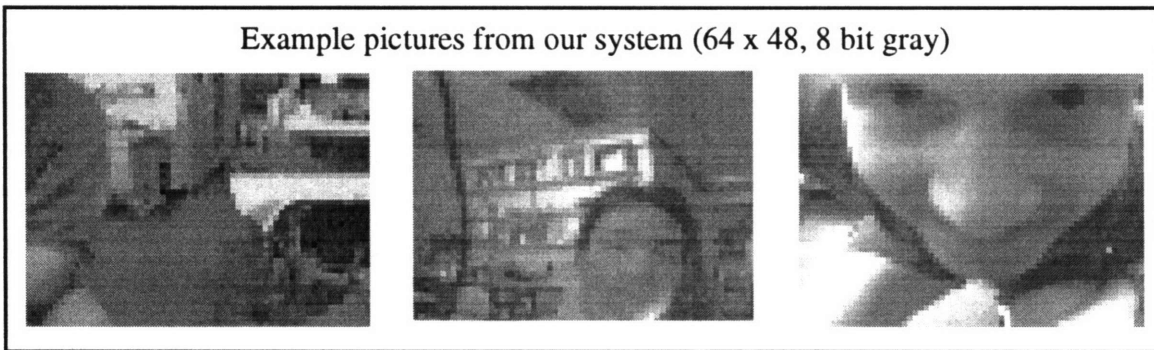
I have equipped the Mobot Vision System with a relatively wide angle lens (approximately 60 to 70 degrees). This is most useful for robot applications because the relative characteristics of space and objects around the robot are of more concern than the specific details at any one point.

Human perception of these images is quite good (see example images). Objects approximately a meter square are easily visible at 20 feet. When angled down toward the ground the camera gives a good view of the space into which a forward driving robot will soon be entering.

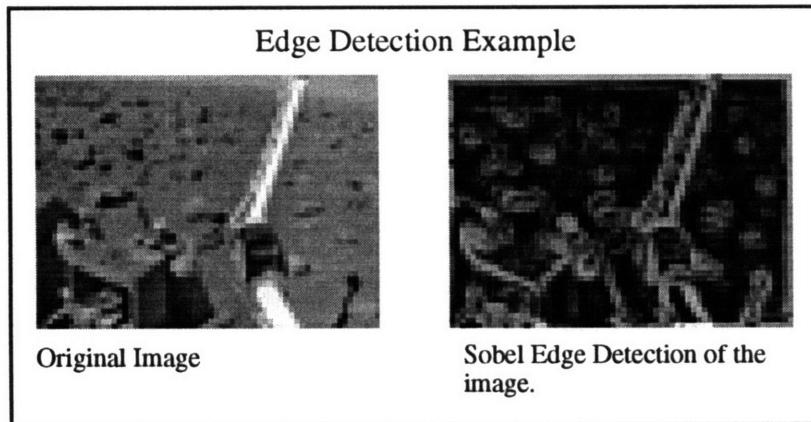
When all of the Vision Board variable knobs (adjustable delay, signal offset knob, and signal gain knob) are tuned properly the system captures an image with a full range of gray scales, which means a smooth clear image to the human eye. These parameters, while interrelated, can be tuned to a specific instance of the system in a few minutes. There is then little need to deal with them again unless a major system component (such as the amplifier, analog switch, or CCD) is exchanged.

The system is however fairly sensitive to light level. The CCD is very sensitive and requires at least a 10% pass filter to operate at normal light levels. In bright sunlight I

usually add an additional 33% pass filter. In order to deal with a greater range of light levels I have built gain control into the frame grabbing software. This routine calculates the average intensity of the frame and adjusts the integration time in a negative feedback loop. The intent is to keep the image at the desired brightness level. This routine extends the dynamic range by about ten decibels, sufficient to encompass most operating conditions in an indoor environment. At some extremes of this range the image becomes more highly contrasting and less smooth. Another drawback of the software gain control is that in low light levels the frame rate is dropped in order to increase the integration time.



The 10 db dynamic range is not sufficient to cover multiple environmental extremes, for example the difference between outside under sunlight and nighttime. To deal with this situation additional hardware would be required to increase the capabilities of the system. We have considered several other options for future improvement of the system. Currently I manually change the filters on the occasion that the lighting conditions require it (usually only when the robot is moved from indoors to outdoors or vice versa). However, some form of automatic solution would be much better. One possible solution might be a filter pinwheel, while another would be the kind of self adjusting filter found in some sunglasses. Additionally we are exploring the possibility of using a CCD with an electronic shutter.



This is the Sobel operator, and it can be seen applied in the above figure. The square-root is not really necessary for most simple vision tasks, and so the function shown below works quite well:

$$|S| = \Delta X + \Delta Y$$

$$\Delta X = |pixel - left(pixel)|$$

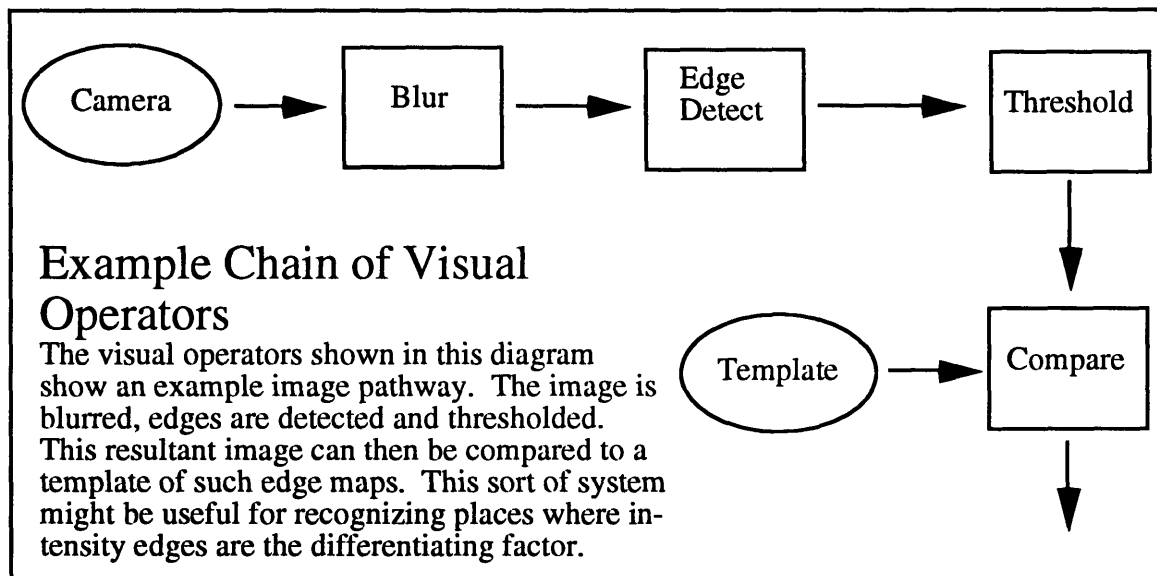
$$\Delta Y = |pixel - up(pixel)|$$

Multiplication and square-root are both high cost operations on simple processors, and should be avoided when attempting to do realtime work on them. This operator will yield an intensity value in the new image that is the magnitude of the horizontal and vertical edges in the original edges. Locally this consists of the amount which a pixel differs from the pixels above and to the left of it. The Sobel edge function is essentially a distance computation, while the simplified version is a calculation of the Manhattan distance. Typically the result on an edge detection is thresholded in order to find edges of a meaningful magnitude.

The final image operator I will discuss is the most complex: *optical flow* (11 instructions per pixel including one multiply) [Horn and Schunck 81]. Optical flow requires at

“pixel to the left,” “pixel above,” and “pixel below” can be calculated by adding the respective offsets: 1, -1, -64, and 64 to the current pixel pointer. This simplification fails for the pixels on the border of the image, in general it is much easier to just ignore the border areas of the image.

The output of image operators can be conceptually connected to the input of other operators. Note however that image operators produce only other images, and there is another class of operators which take images as input, and produce one dimensional arrays, points, and other smaller outputs. In connecting these operators the most common strategy is to run the camera image through a chain of operators in order to extract the required information. Typically this pathway is executed once per frame, but this is not a strict requirement, some operators even require images from different time slices.



Perhaps the simplest image operator is *thresholding* (4 instructions per pixel)¹, which

1. For most of the visual operators presented here I have indicated the cost per pixel in 68332 assembly code. This gives a rough measure of computational demands. Even at the small image size I work with the

merely consists of iterating through an image and producing a new binary image whose value is true whenever a simple function of the input image is true. The simplest form of this is an equation such as:

$$out = (in > threshold)?255:0$$

Another extremely simple operator is *blurring* (11 instructions per pixel). This process is mathematically equivalent to defocusing a lens. Blurring is effectively a low pass filter because it removes high frequency features such as single pixels. This operation generates a new image, a blur of the old one by averaging pixels with those around it. Traditionally a Gaussian filter is applied continuously to the image (a process called convolution), however, for low cost vision nearly identical results can be achieved much more efficiently by making each new pixel an average consisting of four parts the original pixel and one part each the pixels in the cardinal directions.

$$blur = (in * 4 + left(in) + right(in) + up(in) + down(in)) / 8$$

The complement of the blur is the center/surround *sharpen* (11 instructions per pixel). This technique is very similar to the process employed by the retina. In the retina a secondary layer of neurons is excited by a small group of photoreceptors, and inhibited by a surrounding circle. This acts as a high pass filter, reducing low frequency information and boosting high frequency information. While traditionally sharpen is implemented by applying a sombrero shaped curve to the image it is more easily implemented by making the output pixel consist of the input pixel times four, minus each of the surrounding cardinal

looping primitives are reasonable when processing an image: 1 instruction per pixel. However, in most cases several visual operators can be combined into one pass through the image, sparing both memory and computation. The values specified here are for the actual operation on one pixel only and do not include the loop overhead.

pixels.

$$\text{sharpen} = (\text{in} * 4) - \text{left}(\text{in}) - \text{right}(\text{in}) - \text{up}(\text{in}) - \text{down}(\text{in})$$

An extremely simple but useful operator is the *difference* (1 instruction per pixel) operator:

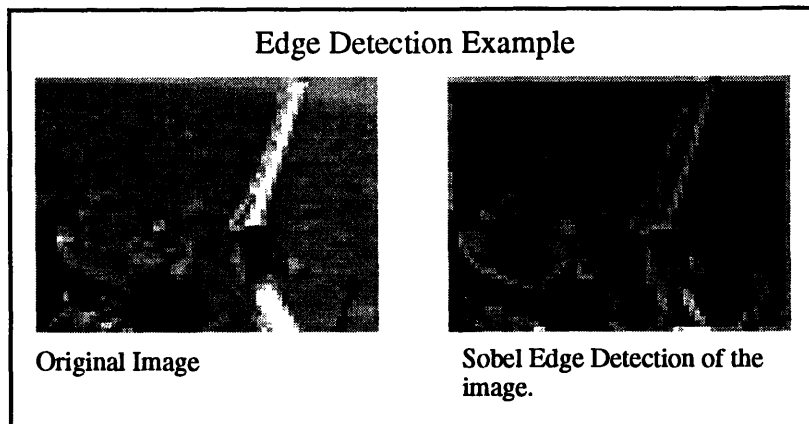
$$\text{out} = |\text{image2} - \text{image1}|$$

This operator gives a measure of the change between one frame and another. When thresholded and passed to the centroid operator (see below) it can be used to locate the center of changed areas of the image.

Edge detection (6 instructions per pixel) is another useful visual operator. Intuitively it consists of finding areas in the image where there are steep intensity transitions. There is an edge gradient, which has a magnitude and direction at each pixel. This gradient can be viewed as the vector perpendicular to the edge at each point, and having a magnitude corresponding to the steepness of the edge. Essentially it points “down the steepest hill” in a 3D plot of the image, where intensity is height. However, for simple realtime vision the magnitude of this gradient is more than sufficient.

$$|S| = \sqrt{S_x^2 + S_y^2} \quad [\text{Neuvatia 82}]$$

S is the edge gradient



This is the Sobel operator, and it can be seen applied in the above figure. The square-root is not really necessary for most simple vision tasks, and so the function shown below works quite well:

$$|S| = \Delta X + \Delta Y$$

$$\Delta X = |pixel - left(pixel)|$$

$$\Delta Y = |pixel - up(pixel)|$$

Multiplication and square-root are both high cost operations on simple processors, and should be avoided when attempting to do realtime work on them. This operator will yield an intensity value in the new image that is the magnitude of the horizontal and vertical edges in the original edges. Locally this consists of the amount which a pixel differs from the pixels above and to the left of it. The Sobel edge function is essentially a distance computation, while the simplified version is a calculation of the Manhattan distance. Typically the result on an edge detection is thresholded in order to find edges of a meaningful magnitude.

The final image operator I will discuss is the most complex: *optical flow* (11 instructions per pixel including one multiply) [Horn and Schunck 81]. Optical flow requires at

least two frames adjacent in time. It consists of another gradient, the direction of which indicates where the intensity of the image is flowing over time, while the magnitude is how fast. Again, the magnitude of this gradient is often sufficient for many applications, and is vastly easier to calculate:

$$|F| = \frac{-F_t}{\sqrt{F_x^2 + F_y^2}} \quad \text{[Ballard and Brown 82]}$$

F is the flow gradient

In a real image the first order differentials of the gradient in any of its three dimensions (x, y or t) can be calculated by subtracting adjacent pixels in that direction. I have managed to develop a realtime version of this procedure by precalculating a fixed point table for the function:

$$\frac{1}{\sqrt{\Delta X^2 + \Delta Y^2}}$$

$$\Delta X = |pixel - left(pixel)|$$

$$\Delta Y = |pixel - up(pixel)|$$

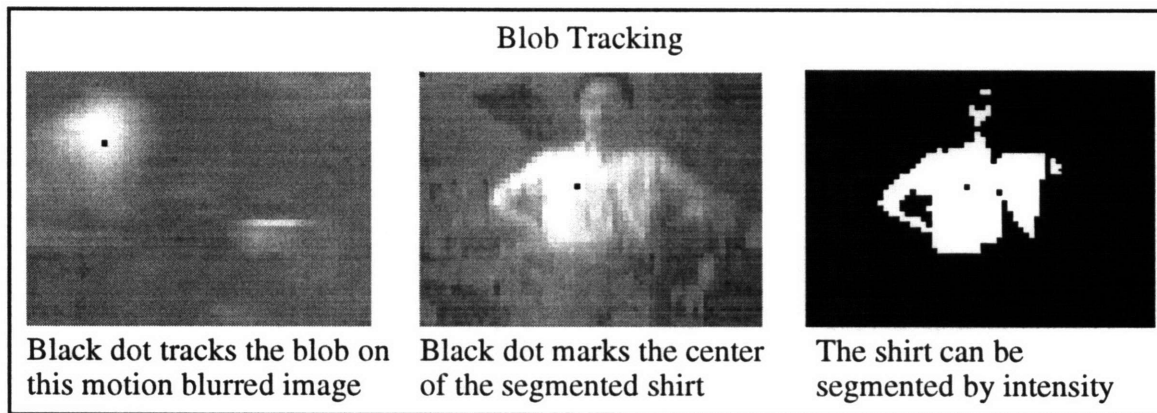
$$\Delta T = |pixel - previous_frame(pixel)|$$

I then feed the table two eight bit deltas and multiply the result by $-\Delta T$ to closely approximate the true flow magnitude described above. The results from this are accurate within the range of the fixed point representation I use, and allow for optical flow magnitude calculations on my board in excess of 30 hertz.

Simple vision is an attempt to extract some small information from a stream of input images. The operators presented above transform images into other images, but do not reduce the size of the data. Below I will discuss a number of simple tools to reduce the 2D

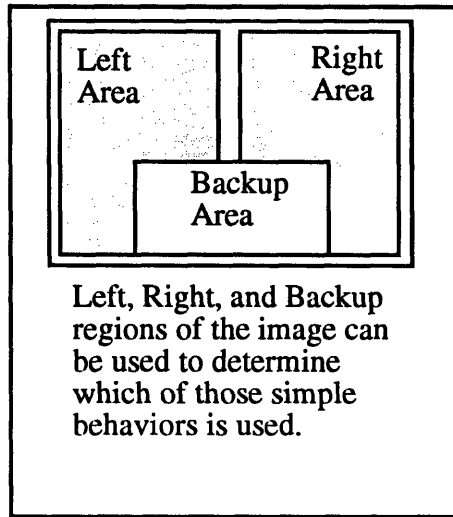
array to a one-dimensional one, and then to other simpler forms.

The first such operator is *centroid* (1 instruction for a black pixel, 4 for a white one, can be combined with thresholding at only a single instruction per pixel cost), which takes as input a binary image, and produces a point. The calculation of a centroid is another simple process. It is a weighted average of the white areas. Essentially the X and Y values of all white pixels are summed independently, and then divided by the number of white pixels, achieving an average location for these pixels. An example of thresholding for intensity and the location of the centroid can be seen below. Intuitively the centroid will find the center of a single blob, or the weighted center of a number of blobs.



Another useful operator is *sum_rect* (1 instruction per pixel), which sums the intensity of all the pixels in a rectangular subset of the image. Essentially it just iterates through this rectangle and adds the pixels.

$$sum_rect(rect) = \sum_{x_1 \rightarrow x_2, y_1 \rightarrow y_2} pixel$$



By calling this routine on subsets of images produced by an operator sequence it is possible to get very useful information. For example, to the left is a breakdown of the screen into three rectangles. One extremely simple navigation system might calculate the optical flow of the camera stream, and then use the summed values of the two side areas to determine which side had more flow. In its simplest form this system could behave as follows:

$$turn = \frac{(sum_rect(right) - sum_rect(left))}{scale}$$

positive turns are to the right, negative to the left

In a rapidly moving robot this system would tend to veer away from objects. In addition, the backup area could be used to add a panic mode to the system. If $sum_rect(backup) > threshold$, then the robot could be placed into emergency mode where it backs up a short distance. Sum_rect can also be used to find the average intensity of a subset of the image (or the entire image) by dividing its result by the number of pixels in the region.

Another simple operator is *find_distances* (1 instruction per pixel below the first white pixel in each column, zero above this pixel), extracts a one dimensional array (called a *distance vector*) from the image with as many elements as the width of the image. Each element of the new array is basically the height in the image of the first white pixel in that column. A low height indicates white in the bottom of that column, a high height no white, or white near the top. This function can be used to convert a binary image where white

represents objects, and black ground, into a one dimensional depth map. This tool allows one to take a binary image separating figure and ground and convert it into a column based depth map, vastly reducing the complexity of the data. This transformation relies on the *ground plane constraint*, using the assumption that the ground is flat to reason that figure high in the image is farther away. In addition, the task of constructing a series of image operators to separate figure and ground is non-trivial.

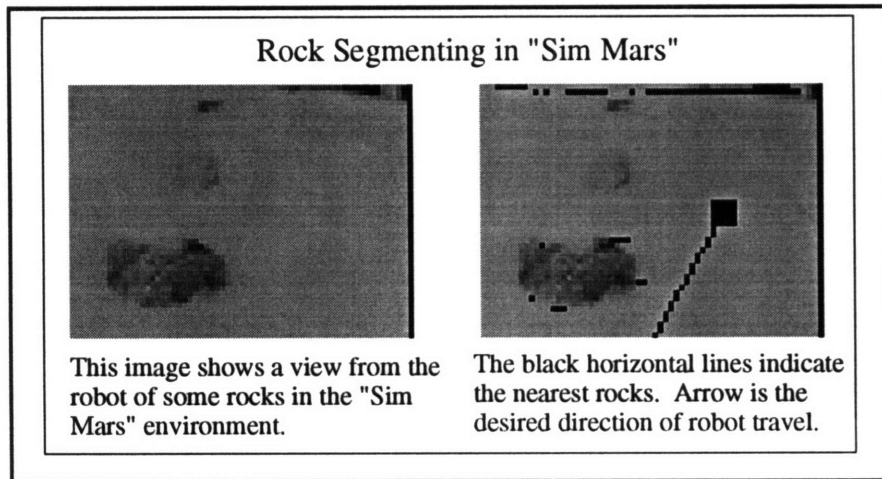
Ian Horswill and I (as the two individuals programming this vision system) have created a number of other simple operators to extract even more refined information from the distance vectors produced by `find_distances`. *Region_min* (2-4 instructions per column)² searches a subset of the vector for the smallest distance in that vector and returns it. This function can be used in a manner similar to `sum_rect` to extract directional information. By dividing the vector into left and right halves:

$$turn = \frac{region_min(right) - region_min(left)}{scale}$$

This is the basis of the control system for two of the robots employing my vision system: Frankie [Horswill and Yamamoto 93] and the Gopher robot in its early forms [Gavin and Yamamoto 93].

Another useful operator is *vector_min* (2-4 instructions per column), which finds the number of the column with the shortest distance in a subset of a distance vector. This can be used to find the horizontal location of the nearest obstacle.

2. It should be noted that it is extremely computationally advantageous to reduce the data pathway from image to vector. The size of the average iteration goes from 3072 on my board to 64 (the width of the image). This is a reduction of 48 times. Once the data stream has made this transition the need to be concerned with assembly level computations is gone.



I have also created a series of operators for doing more extensive analysis on distance vectors. The first of these operators, *seg_vector* (4-6 instructions per column) essentially pre-

forms a run length encoding of the distance vector. It produces a new *segment vector* which is a list of segments of adjacent columns which have the same value within a specified tolerance. Another call, *max_segment* (4-6 instructions per segment)³ can be used to find the horizontal position of the segment which is farthest away and which is larger than a certain threshold. This threshold essentially indicates the maximum width of a "passage" between obstacles which the robot might consider transversing. The call will chose the location of the "deepest traversable passage" from which the turn value can be trivially calculated.

The operators described above provide a toolbox of various low cost vision routines. By careful analysis of the constraints of an environment it is possible to construct a realtime algorithm which uses a series of these or similar operators to extract useful information from the image stream.

3. Computations segments are even faster then on vectors. A segment has perhaps an average of 2-5 times fewer elements than a vector in a typical image.

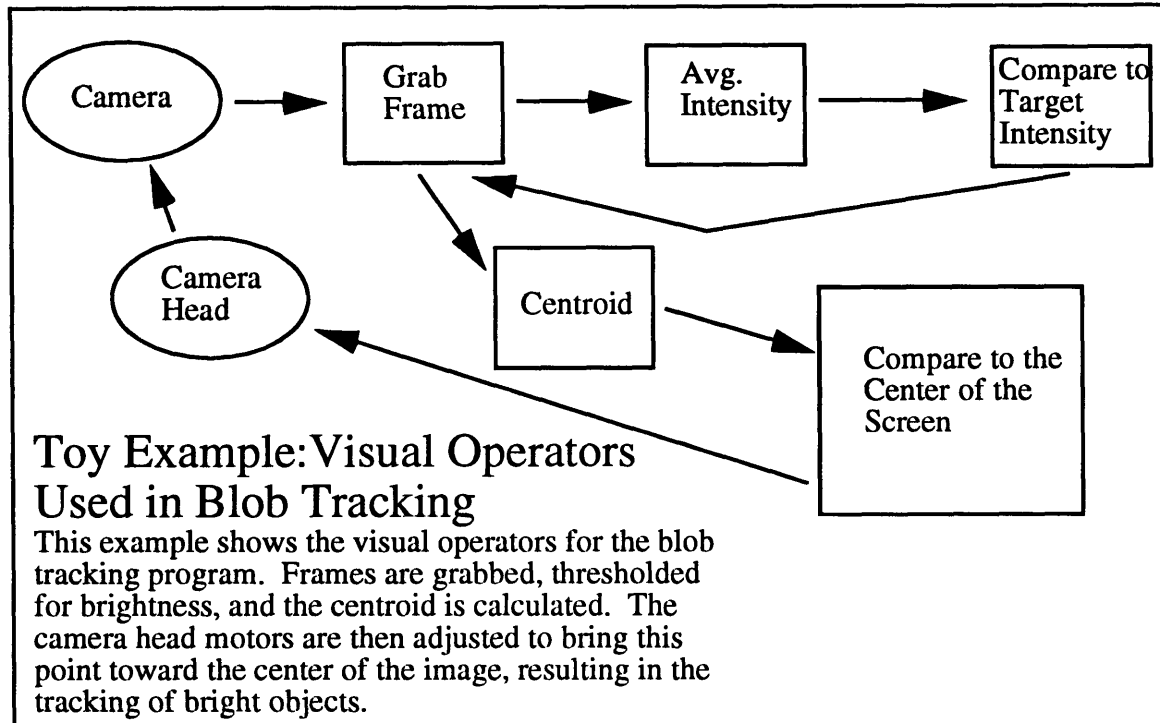
5.3 Software System

The Mobot Vision System is programmable in either 68332 (680x0) assembly or in C using the Gnu C Compiler (GCC). I have written a variety of basic routines. These setup the system, grab frames, actively adjust the integration time based on image light levels, output video frames, and move the camera via two servo actuators.

My C system relies on an excellent 68332 library written by Anne Wright, an MIT undergraduate. This library provides standard I/O routines for C on that processor, as well as numerous calls to control the 68332's many internal functions. On top of this library I have written a series of device drivers to control the Mobot Vision System. These drivers allow the grabbing of frames as images with or without software gain control, the output of frames to the video display hardware at software selectable scale, the relative and absolute positioning of the camera head, and the control of various other board I/O features.

In addition to the vision routines I have written a number of calls to copy and manipulate images, as well as overlay lines, circles, boxes, cross-hairs, arrows, graphs, segments and vectors on images. These features are generally useful in making sense of the visual data. They enable me to plot histograms of the image, and to overlay an arrow indicating the direction that the robot desires to travel on what it sees.

There are also a series of interface routines for grabbing and displaying individual and groups of frames. The system can be used to capture a short film (about 10-15 seconds fit in memory) which can be stored for later playback: forward, backwards, and at different rates. The same set of operators that can be applied live can also be applied to these stored frames so that it is possible to create short films of processed sequences. There also also calls used to transfer images from the vision board to the development computer via the serial port.



I have also written a number of miscellaneous vision routines which are not used directly to guide the robot. These include a routine to input two images and locate an areas that have changed between the images (displayed with a dot indicating the center of the changed area), and a routine to track the brightest blob in the image.

5.4 Computational Cost of Simple Vision on the Mobot Vision System

The 68332 in the Mobot Vision System runs at 16 mhz (actually it is software controlled and can be set anywhere from a low power stop to about 20 mhz, after which it overheats). This CPU has no cache and very little pipe-lining, and the average 32 bit instruction (except for multiply, divide, and certain kinds of shifts) takes about 8 cycles.

This means that there are around 2 million instructions per second, if one avoids the 70 cycle multiply and the 100+ cycle divide.

On the Mobot Vision System it takes approximately 7 milliseconds to grab a frame. This means that 10 frames per second worth of frame grabbing occupies 7% of the CPU, leaving 186,000 instructions per frame, or 62 assembly instructions per pixel free at this frame rate. We have efficiently coded a number of basic vision primitives. For a 64 by 48 8 bit image they have the following approximate costs: Blur (11 instructions per pixel), Center/Surround (11 instructions per pixel), Sobel edge detection (6 instruction per pixel), image difference (1 instruction per pixel), Threshold and find centroid (worst case 6 instructions per pixel), Optical Flow Magnitude (11 instructions per pixel).

As can be seen from the above figures a number of these basic operators can easily be combined to make a fairly sophisticated realtime (10 fps) visual algorithm. By making assumptions about the environment it is possible to construct algorithms that do useful work based on vision. For example, as a simple test case I have implemented code that thresholds an image and finds the center of any bright "blobs" in the image (see example images). This code requires at worst case 6 instructions per pixel plus some trivial constant overhead. I then use this information to center the camera on the "brightness center" of the image. The result is that the camera will actively track a person in a white shirt, or holding a flashlight. It is able to do so at 15 hertz while outputting video. This might not seem very useful, but by changing the thresholding conditions the camera would be able to track anything that can be segmented with a small amount of processing. Intensity bands, the optical flow normal, thresholded edges, (and with filters) colored, laser, and infrared objects are all easy candidates for this technique.

We have used the Mobot Vision System to implement a host of other visual based behaviors. Here at MIT Ian Horswill has built Polly, a completely visually guided robot that uses an identical 64 by 48 gray scale image and a processor only somewhat more powerful than the 68332. Polly is designed to give “brain damaged tours of the lab.” It is capable of following corridors, avoiding obstacles, recognizing places, and detecting the presence of people [Horswill 92a] [Horswill 93]. We have brought many of these skills to the Gopher system, and to Frankie, another lab robot based on the Mobot Vision System. The algorithms for avoiding obstacles and following corridors are easily within the power of one of these vision systems. Ian Horswill has also used one of these vision systems, slightly modified by Masaki Yamamoto to add a second camera, to do binocular gaze fixation [Horswill and Yamamoto 93].

5.5 Applications to Mars

NASA is planning on sending a small autonomous rover to Mars in the next few years as part of the Mesur Pathfinder mission. It is our belief that this rover could benefit from some vision based guidance, and that the approach used in the Mobot Vision System would be very suitable. A remote rover sent to Mars will be very limited by weight, size, and power consumption. The current rover design uses a digital CCD camera system quite similar to ours, as well as a low power processor. For power reasons the rover operates at very slow speeds, and so the algorithms we discussed here could run at a reasonable rate on its small processor.

The surface of mars as captured by the Viking Landers is fairly flat (at least where they landed) and regular (see Viking image above). It consists of a surface of tiny dust particles littered with an fairly Gaussian distribution of rocks. A rover’s physical

characteristics will determine what size of rocks are hazards and which are not. It would be useful for a vision system to be able to look forward and estimate roughly how much space in each direction there is until rocks that are too large to cross.

A typical approach to using low cost vision for navigation involves three separate problems:

- The determination within the image of safe and hazardous zones.
- The determination of the relationship between image areas and a robot relative physical geometry. In low cost vision it is not necessary to build a complex geometric model of the world, rather it is sufficient to construct some notion of the relative position of hazards with regard to the robot. This construction of the complete map is extremely costly, and probably not at all necessary.
- The selection of the best course of travel based on the above information.

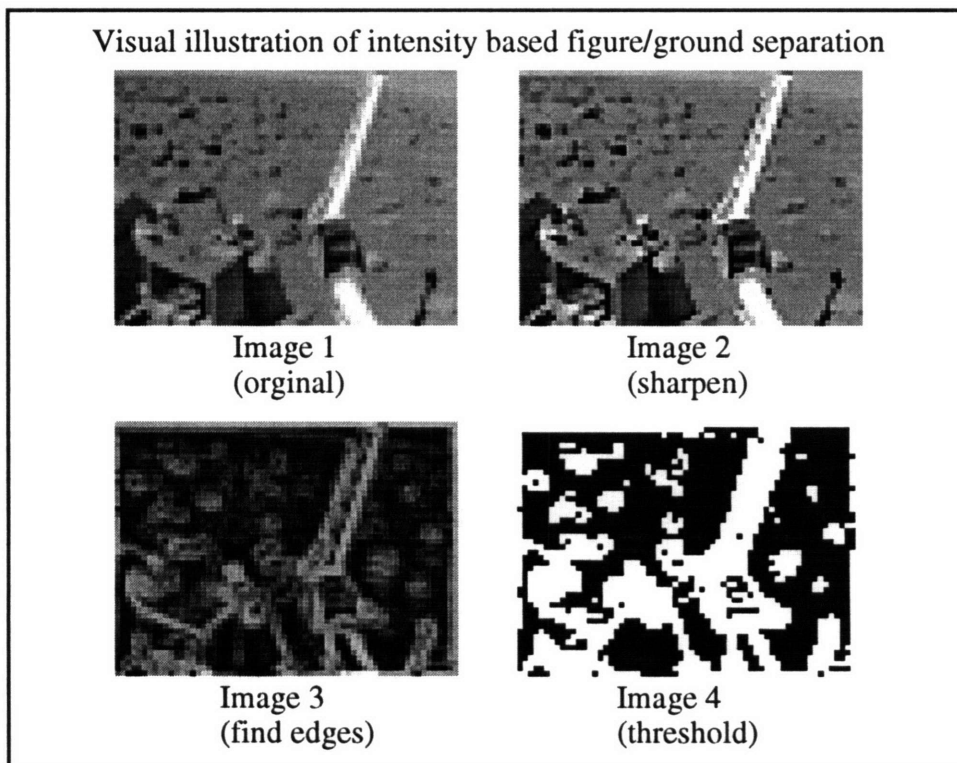
I will show how low cost vision solutions make all of these tasks much easier than they have traditionally been. However, in order to solve these problems cheaply assumptions must be made about the environment. These assumptions, based on constraints in the environment, will allow the construction of simple computations.

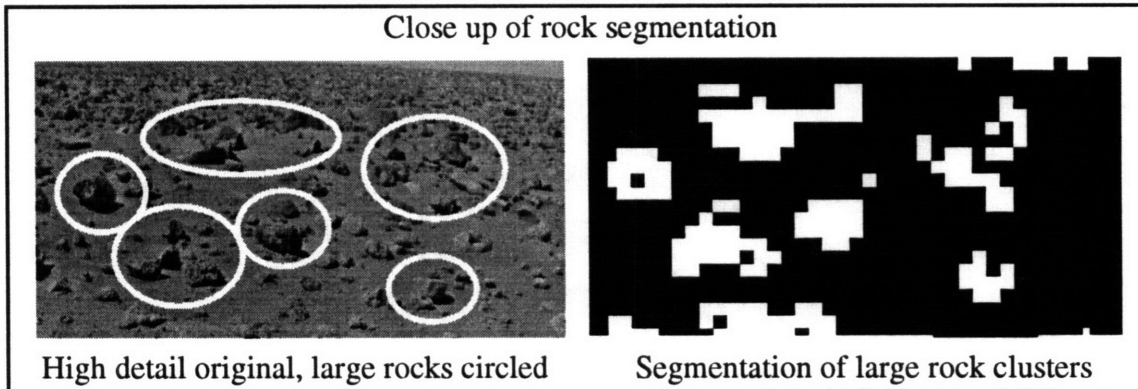
The first of my algorithms is texture based. This algorithm depends on a number of assumptions. These assumptions make it possible to extract useful data from an image in a reasonable time, however, it is important to be aware of the limitations they impose.

- Assumption 1: We wish to determine visually which areas of the image are large rocks or dark pits (figure), and which areas are normal dust and dirt (ground). The dirt and dust is very fine in size, and fairly uniform in color, while the rocks consist of larger contrasting blobs, and pits will be in shadow.
- Result 1: By looking for strong low frequency edges in the image we can locate shadow borders, and larger rocks. This may be adjusted to ignore objects whose frequency is so high as to indicate that they are either too small or are too far away to concern the robot.
- Assumption 2: The local ground area visible to the robot is reasonably flat (the ground plane constraint), and obstacles rest on the ground.

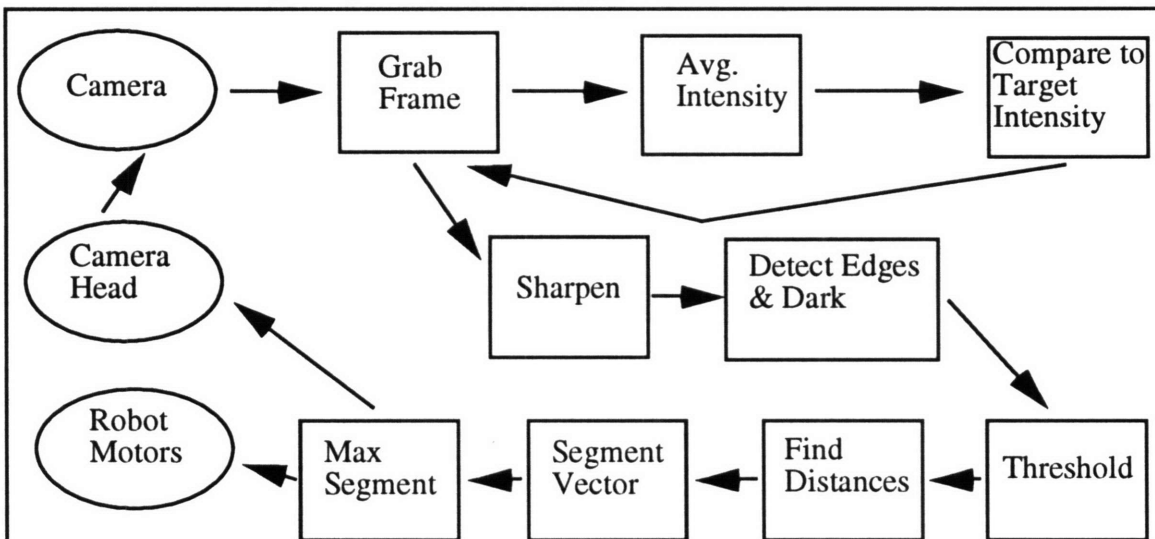
- Result 2: Because obstacles rest on the ground, and the ground is reasonably flat, objects higher in the image will always be farther away than objects lower in the image. We can therefore determine the relative distance of the first obstacle in each column by looking for the lowest pixel of figure in that column.
- Assumption 3: Since we are sampling the world at a rapid rate relative to the speed of travel (20-30 fps at a rate of 1 meter per second) we need to only determine the desired angle of turn. The depth map calculated from the image columns is sufficient to find the relative depth and position of nearby safe regions in front of the robot.
- Result 3: By searching the depth map for the deepest pathway that is wider than a certain threshold we can determine an appropriate direction to turn during this frame. The repeating nature of this process will compensate for transient errors and lack of calibration.

If we assume that the ground is roughly flat, as indeed it is in the Viking images, then rocks that are farther away will be higher up in the image. Additionally the low resolution of the camera is convenient because it will filter the dust and all small rocks into a uniform lack of texture. Texture, and therefore edges, is an indicator of objects or rocks.





If the rocks can be segmented, then by starting at the bottom of the image and looking upward for “rocks” we can create a monotonic depth map of the distance to rocks in various directions. This is a simplified approach because it assumes that the ground is pretty flat. However it works surprising well on the Viking test image.

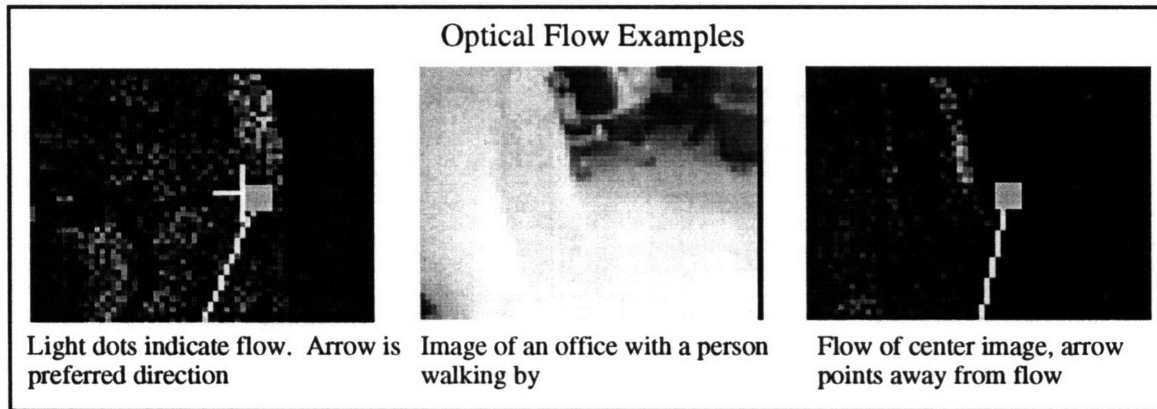


Operators used in Intensity based Navigation

This diagram represents an operator summary of the intensity based algorithm illustrated visually above. Images from the camera are sharpened, edged detected, and then thresholded. Free space is estimated by searching upwards in the image for texture (which signals objects). The free space map is segmented, and then searched for large free areas into which the robot is directed.

Observe the series of images above. Number one is a 64 by 48 image as seen from the Viking Lander. Ignore the lander itself in the image, and assume a rover based camera can be mounted in the front where the rover itself will not be visible. In image 2 we have sharpened the edges of the images, and in image 3 we have used a simple edge detector. Finally image 4 was made by using an intensity threshold. Notice that the larger rocks in the original image are visibly segmented in the final image. By adjusting the threshold it is possible to segment for rocks of various sizes. This method is based on one devised by Horswill [Horswill 92b] and runs in realtime on the Mobot Vision System with processing to spare. We have used this system in several robots to avoid obstacles on flat uniform floors (usually carpets or cement). It works quite robustly.

I have designed a more elaborate version of the algorithm for use in my simulated Mars area. This implementation which looks for dark areas as well as edges as indicators of undesirable areas. This new version uses the more elaborate segmentation system described above under visual operators. This system allows it to choose the deepest path that is wide enough to accommodate itself. In addition this system directs the camera head to look in the direction where the largest free space is. Since the camera head moves more rapidly than the robot itself this serves as a kind of look ahead, directing the robot's attention to the area into which it will soon be traveling. This system has performed well in the simulated Mars environment.

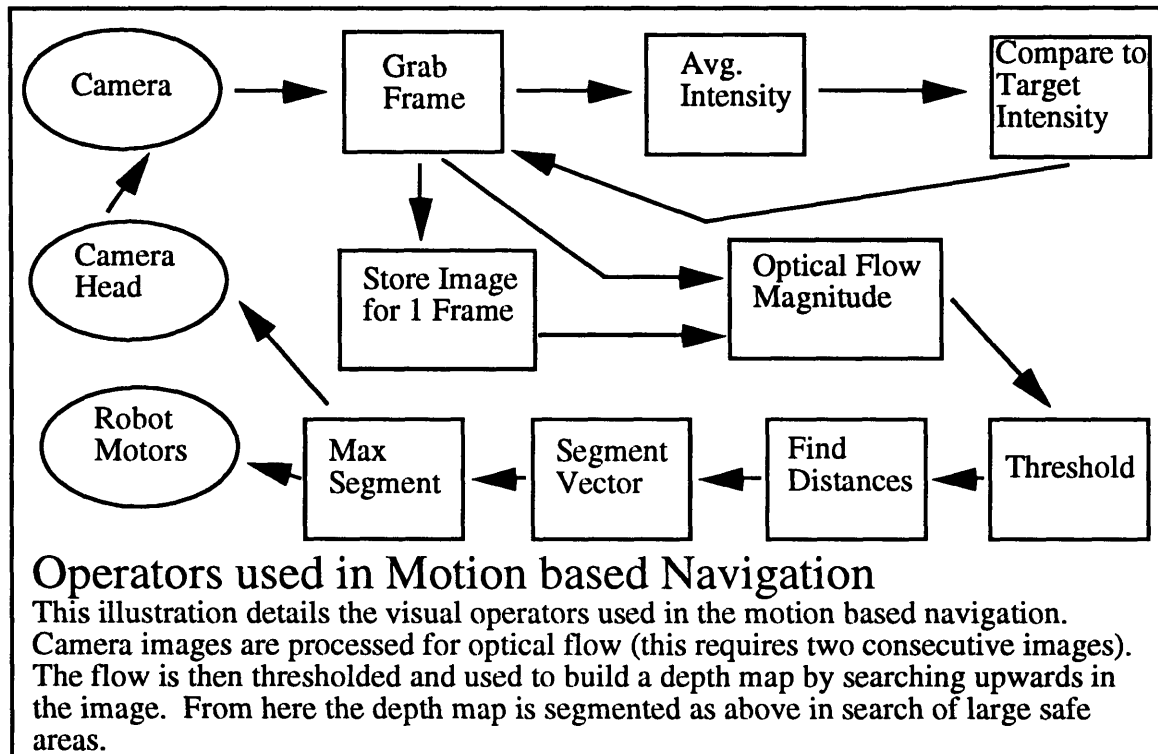


The second algorithm is based on motion.

- Assumption 1: The local ground area visible to the robot is reasonably flat (the ground plane constraint), and obstacles rest on the ground. Since the ground is flat, rocks will protrude from the ground plane.
- Result 1: Because obstacles rest on the ground, and the ground is reasonably flat, objects higher in the image will always be farther away than objects lower in the image. We can therefore determine the relative distance of the first obstacle in each column by looking for the lowest pixel of figure in that column.
- Assumption 2: The robot moves forward. Texture in the image will flow from the edges of the image into the optical center, which represents the direction of travel in the image plane. Objects in the world will move at a speed proportional to $1/d$ where d is the distance of the object from the camera.
- Result 2: Because the rocks protrude from the ground plane, and because they tend to have more texture than the dirt, the optical flow magnitude of the image will highlight the moving rocks, and moreover will highlight them in proportion to the speed at which they are moving. Since closer objects move rapidly, they are more readily seen.
- Assumption 3: Since we are sampling the world at a rapid rate relative to the speed of travel (20-30 fps at a rate of 1 meter per second) we need to only determine the desired angle of turn. The depth map calculated from the image columns is sufficient to find the relative depth and position of nearby safe regions in front of the robot.
- Result 3: By searching the depth map for the deepest pathway that is wider than a certain threshold we can determine an appropriate direction to turn during this frame. The repeating nature of this process will compensate for transient errors and lack of calibration.

I have implemented an operator that calculates the magnitude of the optical flow in the direction of the intensity gradient in realtime. If we make the assumption that the robot is moving forward roughly in the direction of the camera than the rate of motion of an obstacle is proportional to $1/d$ where d is the distance of the obstacle from the camera. This means that there will be very little movement in the center of the direction of travel, and more on the edges. Objects will accelerate rapidly as they approach the camera [Brooks, Flynn, and Marill 87]. This large movement can be seen as increased flow, and with thresholding nearby obstacles can be loosely isolated. An even simpler strategy is to turn the robot in such a manner as to balance the flow on either side the robot. If there is more flow on the left go right, and vice versa. A large amount of flow in the lower area of the image indicates an rapidly looming object. This technique is very similar to that employed by a number of flying insects (see above) [Srinivasan, Lehrer, Kirchner, and Zhang 91]. Balancing flow works well in a moving agent to avoid static objects. Some examples of this in action can be seen here. The line with the square on the end is an arrow indicating the direction the robot should go, the cross indicates an estimate of the direction of motion.

I have also implemented a more complex version of this system that is similar in structure to the intensity based algorithm. In this approach the image is search from the bottom for significant levels of optical flow. This depth map is then processed into segments in order to find the deepest path of acceptable width.

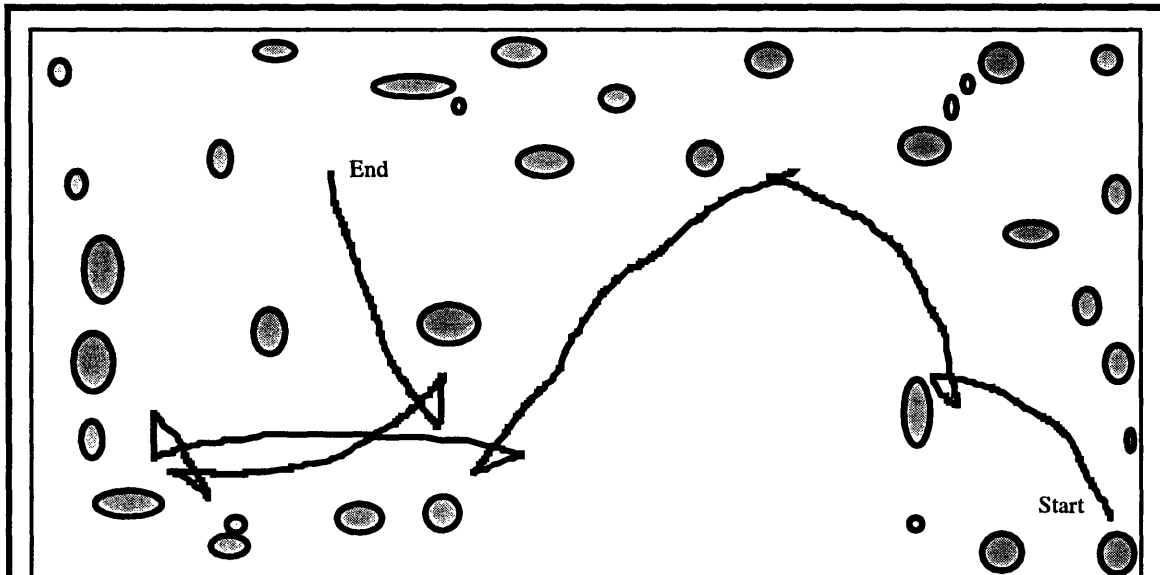


Because our algorithms run in real time, it is not necessary to convert to a complex three dimensional map of the world. We can convert straight from an image based map to a simple robot relative map. This is actually much more useful to a robot, and is vastly less computationally intensive. The realtime nature of our calculations applies a temporal smoothing to any errors made by the algorithm. This type of vision calculation tends to be very noisy, but with temporal smoothing, this noise is greatly reduced without have to resort to very difficult and time consuming calculations.

Chapter 6

Results

I have used the Mobot Vision System and the algorithms described above to guide the Gopher robot successfully through a number of environments. At the AAAI conference in July of 1993 Gopher was run successfully for several days with an early version of the intensity based algorithm. This included several different environments, such as concrete floor and carpeted floor. For the most part the robot was able to avoid colliding with any obstacles, instead veering away from them. For a number of afternoons it was allowed to wander a crowded exhibition area. It avoided colliding with the humans, walls, and tables. Besides the vision system, Gopher is only equipped with a touch strip (which can sense fairly hard contact along the the base of the robot) and infrared sensors which can detect objects with high infrared reflectivity at about six inches. Without the vision, it could not have avoided bumping into the human occupants found in the exhibition hall.



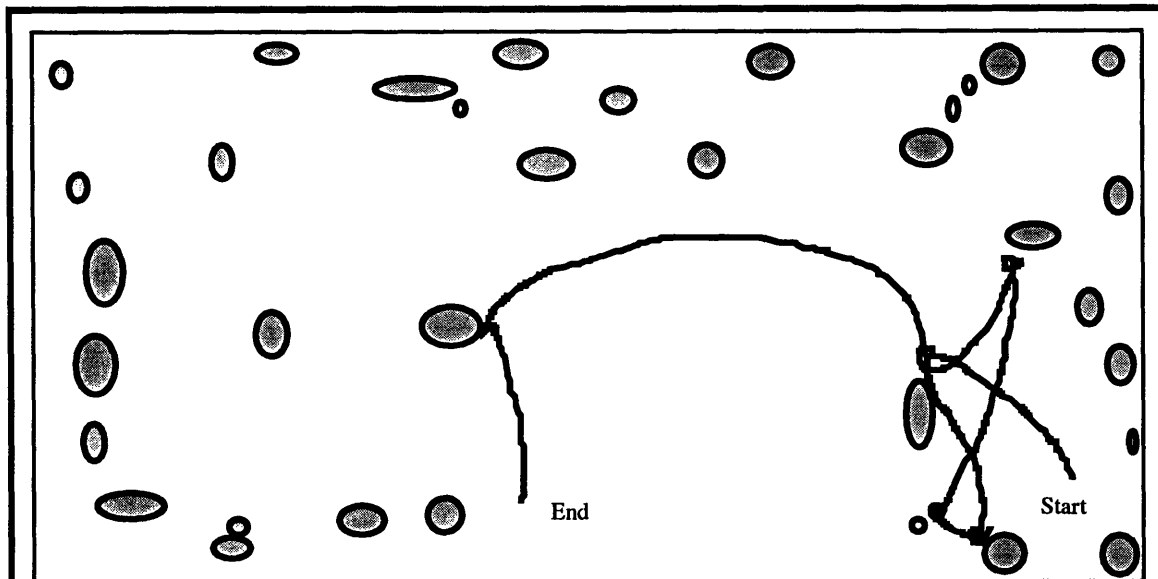
A two minute example run using vision in Sim Mars

This illustration shows the path of the robot during a two minute vision guided run in the simulated Mars environment. The robot was using the intensity based algorithm described above. The "start" and "end" points are marked. Bow ties in the path are caused when the robot detected an obstacle, and backed away in order to avoid it.

Gopher was also able to run quite well on the seventh floor of the MIT Artificial Intelligence Lab. This is the environment that its algorithmic ancestor, Polly, was designed to operate in. Like that robot Gopher is able to avoid walls, chairs, piles of junk, and even dynamically relocated people (which it of course views as feet and legs).

Since it was my goal to create a robot vision system useful for a Martian rover I have run the Robot in the simulated Mars environment under a number of algorithms. Presented here are tracings of two trial runs. In the first of these the robot is running under the more sophisticated version of the intensity based algorithm. The robot is undirected except for its local desire to turn toward the area of greatest perceived free space. Each of these runs had a duration of two minutes.

Gopher's wheel base is severely misaligned and so the robot tends to arc mildly as it attempts to drive straight. Nevertheless, Gopher can be seen turn away from rocks significantly before hitting them. The robot's typical behavior is to turn away from rocks that enter its peripheral vision (not very far to the side given that Gopher's camera has only a 90 degree view of field) and to stop and back away from obstacles that it approaches head on. As soon as the robot has backed sufficiently far away it will turn toward the side with the most visible free space and drive that way. Although Gopher is not particularly brilliant or directed vision it to not only avoid the rocks but to find its way out of concave groupings. While it is not systematic in this last behavior, its fairly wide and distant view of the world (on a robot relative scale) combined with the algorithm's innate preference for open areas tends to get the machine out of enclosed spaces much more rapidly than it would without vision.



A two minute example run in Sim Mars without vision

This illustration shows the path of the robot during a two minute run in the simulated Mars environment in which infrared and touch detection were the only available sensors. The "start" and "end" points are marked. Notice how the robot spends a significant amount of time trapped in a confined area, while in the above visually guided run it steered its way free more rapidly. Without vision the robot is able to explore significantly less territory in a given amount of time.

Above can be seen a second trial run in which the robot was run without vision. Since its best chance of avoiding a rock in this mode is to detect it with one of its scattered infrared sensors it approached much closer to the obstacles, often bumping into them (and on occasion dragging the rock). The close range sensors do not provide the robot with any useful information about a desirable new direction in which to head after a collision, and so it tends to circle or get caught in a repeating behavior. In addition the blind robot is able to drive right past an opening in an enclosed space without having any means by which to determine that it should head in that direction. The overall result is that the robot hit more rocks, spent more timing circling, and was often caught in a small area.

Overall the simple vision system and algorithms detailed here are enough to provide useful navigation information to a mobile robot. While far from competing for the next Nobel prize, this extra information brings the robot to a higher level of intelligent behavior, giving it a smoother, less blundering path.

The current algorithm employed by my robot is without any sort of higher level goal, instead seeking only to avoid collisions. I believe that it could be easily extended to include some higher level direction, and provide much more intelligent behavior.

Chapter 7

Conclusions

7.1 Summary

The images sampled by visual sensors, be they organic eyes, or inorganic cameras, contain a great deal of useful information about the environment, often available at more distance than other sensors allow. I believe that current technology makes it possible for a mobile robot to use vision as a guide for well defined tasks. Knowledge of the tasks and constraints in the environment enable the information useful in solving the task to be extracted from the visual input at a reasonable computational cost.

I have constructed a small self contained vision system. It employs a cheap but fast general purpose microcontroller (a 68332) connected to a Charge Coupled Device (CCD). The CCD provides the CPU with a continuous series of medium resolution gray-scale images (64 by 48 pixels with 256 gray levels at 10-30 frames a second). In order to accommodate our goals of low power, light weight, and small size we are bypassing the traditional NTSC video and using a purely digital solution. As the frames are captured any desired algorithm can then be implemented on the microcontroller to extract the desired information from the images and communicate it to the host robot.

I have successfully programmed this system to navigate a mobile robot through a rock field using vision. The vision-based robot follows a more efficient navigation path than the same robot relying only on touch and infrared sensors. It is able to avoid obstacles

at a greater distance, increasing both overall transversal speed and robot safety.

I believe that the limited amount of processing available on our Mobot Vision System, when combined with a low resolution image and reasonable assumptions about the environment are sufficient to provide a robot with a good deal of useful information. I also believe that a small, light, cheap vision system such as this could be applied to many branches of the robotics field, including space exploration, indoor, and outdoor robots.

7.2 Areas for Further Research

While quite useful, the Mobot Vision hardware is first generation. One of its major problems is its limited dynamic range. Illumination levels in the real world vary over a range that considerably exceeds that of current CCD technology, but the human eye and visual system have evolved several sophisticated means for dealing with this problem. The iris provides a physical regulator of light, while at the same time there are two different sets of photoreceptors, tuned to different light levels, and within each of these adaptive chemical systems designed to handle a wide range of illuminations [Kandel, Schwartz, et al. 91]. This multi layered approach could also be applied to an artificial system such as mine. A physical or optical iris could be coupled with an electronically shuttered CCD and software gain control to provide the desired dynamic range.

The vision system could always benefit from more processing power as long as the cost was not too high. Fortunately advances in chip technology produce faster single chip microcontrollers every year. Already the 68332 I use is available in a configuration that is twice as fast as mine at the same size and power level. Additionally, there is a new generation of DSP and other specialized microprocessors that promise to yield large increases in the type of operations performed by low level vision.

Finally there is room for a more advanced video output option. My simple solution provides video at an extremely low chip cost, however the output quality is poor and its drag on CPU resources is significant. Solutions for low resolution frame buffers are readily available today and could be used to make the debugging board more useful.

In terms of testing conditions it would be desirable to situate the vision system in a robot with a base more suitable to the transversal of outdoor terrain. The Rocky base [Wilcox 94] is a good example of this, providing a flexible six wheel base for transversal of rocks and sands. In addition, the testing environment should be improved to provide a more realistic environment. If one was using a robot base capable of crossing sand and small rocks it would be desirable to build a better Mars environment consisting of drifts of sand and more rocks.

However, the most interesting area for further research is the improvement of the vision algorithms. There are many ways in which the implementation I have created could be improved. Firstly one could attempt to improve the reliability of the figure ground separation. The current intensity algorithm is very conservative, avoiding specular reflections and shadows with which it would have no trouble. On the other hand, the flow based algorithm is quite liberal, failing to notice small obstacles. One very interesting approach which I had hoped to employ, but did not have time for, is the integration of multiple algorithms. Multiple techniques which work to solve the same or similar task can in theory be integrated for increased reliability. A simple conservative solution to combining my two figure ground routines would be to use the logical OR of the two. Ideally one would be able to generate some kind of confidence measure for each algorithm

and use the best one. It might even be possible to do this on a sub image scale. If one knows the assumptions that a simple vision algorithm uses, and can detect when these assumptions are being violated, than it would be possible to systematically construct confidence measures.

Both the hardware and software approaches I have detailed here are appropriate for a number of additional solutions to the surface navigation problem, as well as countless other simple vision applications. The hardware already exists in a stereo form, and this opens the door for countless additional simple algorithms. Many problems can be solved in stereo using just one scanline, for example gaze control [Horswill and Yamamoto 93].

Another weakness of my implementation is its lack of high level guidance. The current code merely guides the robot toward the largest free space. However, there are numerous higher level additions to this that are possible. One very simple approach would be to give the robot a preference for a particular compass direction (the Gopher robot has a flux gate compass, but the solar meridian can also be calculated from the brightest area on the horizon). Another approach might be to have the robot fixate on sub goals in the image, track them across frames (done fairly simply by comparing the differences of a small area between frames), and steer the robot preferentially toward these goals. The sub goals could initially be acquired by human intervention (a method which Nasa intends to use on the Pathfinder robot) or under program control. For example, in the case of the Martian rover, it should be possible to construct a simple vision routine to locate the rover's lander if it is in view because the lander differs considerably from the native Martian objects. The rover could then track the lander while it could, and use this to position itself or return to home.

Overall we have only just begun the exploration of the use of simple realtime vision for the guidance of mobile robots. The enormous number of problems which biological systems solve by vision, including some animals with fairly simple brains, leads me to believe that there are enormous gains in robot competence to be made by the intelligent analysis and construction of task oriented vision.

- Angle 89 "Ghenghis, a Six Legged Autonomous Walking Robot", Colin M. Angle, *MIT SB Thesis*, March 1989.
- Ballard and Brown 82 Computer Vision, Dana H. Ballard and Christopher M. Brown, *Prentice-Hall*, 1982, pp. 102-103.
- Brooks 85 "A Robust Layered Control System for a Mobile Robot", Rodney A. Brooks, *A.I. Memo No 864, MIT AI Lab*, 1985.
- Brooks 90 "The Behavior Language; User's Guide", Rodney A. Brooks, *A.I. Memo No 1227, MIT AI Lab*, 1990.
- Brooks 91 "Intelligence Without Reason", Rodney A. Brooks, *A.I. Memo No 1293, MIT AI Lab*, 1991.
- Brooks, Connell and Ning 88 "Herbert: A Second Generation Mobile Robot", Rodney A. Brooks, Jonathan H. Connell and Peter Ning, *A.I. Memo No 1016, MIT AI Lab*, 1988.
- Brooks, Flynn, and Marill 87 "Self Calibration of Motion and Stereo Vision for Mobile Robots", Rodney A. Brooks, Anita M. Flynn and Thomas Marill, *4th Int. Symposium of Robotics Research, Santa Cruz, Ca*, 1987.
- Coombs and Roberts 92 "Centering behavior using peripheral vision", D. Coombs and K. Roberts, In D. P. Casasent, editor, *Intelligent Robots and Computer Vision XI: Algorithms, Techniques and Active Vision*, pages 714-21. SPIE, Vol. 1825, Nov. 1992.
- Desimone, Schein et al. 85 Desimone, R., Schein, S. J., Moran, J., Ungerleider, L. G. 1985 "Contour, Color, and Shape Analysis Beyond the Striate Cortex." *Vision Research*, Vol. 25, No. 3, pp. 441-452.
- Ferrell 93 "Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators", Cynthia Ferrell, *MIT SM Thesis*, 1993.
- Flynn, Brooks, Wells, and Barrett 89 "Squirt: The Prototypical Mobile Robot for Autonomous Graduate Students", Anita M. Flynn, Rodney A. Brooks, William M. Wells III, David S. Barrett, *A. I. Memo 1120, MIT AI Lab*, 1989.
- Gavin 93 "A fast, cheap, and easy vision system for an autonomous vacuum cleaner", Andrew S. Gavin, *AAAI Fall Symposium*, October 1993.

- Gavin and Brooks 94 "Low Computation Vision-Based Navigation for a Martian Rover", Andrew S. Gavin and Rodney A. Brooks, *Proceedings to the Conference on Intelligent Robotics in Field, Factor, Service, and Space*, Vol II, pp. 685-695, 1994.
- Gavin and Yamamoto 93 "A fast, cheap, and easy system for outside vision on Mars", Andrew S. Gavin and Masaki Yamamoto, *Intelligent Robots and Computer Vision XI*, Boston, MA, September 1993. SPIE.
- Goto and Stenz 87 "The CMU System for mobile robot navigation", Y. Goto and A. Stenz, *IEEE International Conference on Robotics and Automation*, pages 99-105. IEEE, March 87.
- Horn and Schunck 81 "Determining optical flow", B. K. P. Horn and B. G. Schunck, *Artificial Intelligence*, 17 No. 1-3, pp. 185-204, 1981.
- Horswill 91 "Characterizing Adaption by Constraint", Ian Horswill, *Proceedings 1991 European Conference on Artificial Life*, MIT Press, 1991.
- Horswill 92a "A simple, cheap, and robust visual navigation system", Ian Horswill, *SAB92*, 1992.
- Horswill 92b "Analysis of Adaption and Environment", Ian Horswill, *AIJ*, 1992.
- Horswill 93 "Specialization of Perception Processes", Ian Horswill, *MIT PhD Thesis*, 1993.
- Horswill and Yamamoto 93 "A \$1000 Active Stereo Vision System", Ian Horswill and Masaki Yamamoto, submitted to *CVPR 94*, 1993.
- JPL 93 Personal notes taken from presentations at Jet Propulsion Laboratories, Pasadena, Ca., August 1993.
- Kandel, Schwartz, et al. 91 Kandel, E. R., Schwartz, J. H., Jessell, T. M. 1991 *Principles of Neural Science*, Third Edition. Elsevier, New York, Amsterdam, London, Tokyo.
- Mishkin, Ungerleider, & Macko 83 Mishkin, M., Ungerleider, L.G., & Macko, K.A. 1983 "Object vision and spatial vision: Two cortical pathways." *Trends in Neuroscience*, 6, pp. 414-417.
- Moravec 82 "The Stanford Cart and the CMU Rover", Hans P. Moravec, *Proceeding of the IEEE*, 71(7), 1982, 872-884.

- Motorola "M68HC11 HCMOS Single-Chip Microcontroller: Programmers Reference Manual", Motorola.
- Motorola 89 "MC68332 SIM System Integration Module: User's Manual", Motorola, 1989.
- Nasa 88 "Environment of Mars" *Nasa Technical Memorandum 100470*, October 1988.
- Nevatia 82 Machine Perception, Ramakant Nevatia, Prentice-Hall, Inc., 1982.
- Nilson 84 "Shakey the Robot", Nils J. Nilsson (ed), *SRI A.I. Center Technical Note 323*, April, 1984.
- Sandini et al 93 "Robotic Bees", G. Sandini, J. Santos-Victor, F. Curotto, and S. Garibaldi, *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, 1993.
- Santschi 11 "Observations et remarques critiques sur le mecanisme de l'orientation chez les fourmis", F. Santschi, *Rev. Suisse Zool.* 19, 303-338.
- Santschi 23 "L'orientation siderale des fourmis, et quelques considerations sur leurs differentes possibilities d'orientation. I. Classification des diverses possibilites d'orientation chez les fourmis", F. Santschi, *Mem. Soc. Vaud. Sci. Nat.* 4, 137-175.
- Srinivasan 92 "Distance perception in insects", M.V. Srinivasan, *Current Directions in Psychological Science* 1, pp. 22-26, 1992.
- Srinivasan, Lehrer, Kirchner, and Zhang 91 "Range perception through apparent image speed in freely flying honeybees", M.V. Srinivasan, M. Lehrer, W.h. Kirchner, and S. W. Zhang, *Visual Neuroscience* 6, pp. 519-535, 1991.
- Storjohann, Zeilke, Mallot, and Seelen 90 "Visual obstacle detection for automatically guided vehicles", *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 761-766, May 1990.
- TI 92 Area Array Image Sensor Products: Data Manual, Texas Instruments, Image Sensor Technology Center, 1992.
- Ungerleider 85 Ungerleider, L. G. 1985 "The Corticocortical Pathways for Object Recognition and Spatial Perception." In *Pattern Recognition Mechanisms* (ed.

C. Chagas, R. Guttass, & C. G. Gross), pp. 179-201. Vatican City: Pontifica Academia Scientiarum & Berlin: Springer-Verlag.

von Frisch 49 "Die Polarisation des Himmelslichts als orientierender Faktor bei den Tänzen der Bienen", K. von Frisch, *Experientia* 5, 142-148.

Wehner 89 "The Hymenopteran Skylight Compass: Matched Filtering and Parallel Coding", Rudiger Wehner, *Journal Exploratory Biology*, vol. 146, pp. 63-85, 1989.

Weisbin 93 "Rover Technology Program", C. R. Weisbin, *Presentation to the DRD*, Jet Propulsion Laboratory, Pasadena, Ca, August 1993.

Wilcox 94 "Non-Geometric Hazard Detection for a Mars Microrover", Brian H. Wilcox, *Proceedings to the Conference on Intelligent Robotics in Field, Factor, Service, and Space*, Vol II, pp. 675-684, 1994.