Detection and Avoidance of Simulated Potholes in Autonomous Vehicle Navigation in an Unstructured Environment

A thesis submitted to the division of
Graduate Studies and Advanced Research
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTERS OF SCIENCE

In the Department of Mechanical, Nuclear and Industrial Engineering
of the College of Engineering
2000

by

By Jaiganesh Karuppuswamy
BE Production Engineering
PSG College of Technology

Thesis Advisor and Committee Chair: Dr. Ernest L. Hall

Abstract

This research discusses a solution to detection and avoidance of simulated potholes in the path of an autonomous vehicle operating in an unstructured environment. Pothole avoidance may be considered similar to other obstacle avoidance except that the potholes are depressions rather than extrusions from a surface. Simulated potholes were used in this research. A vision approach was used since the simulated potholes were significantly different visually from the background surface. Large potholes more than 2 feet in diameter were detected. Furthermore, only white potholes will be detected on a background of grass, asphalt, sand or green painted bridges.

The solution to the problem was developed in a systematic manner. Various approaches to solving the problem were considered. After a specific camera and frame grabber were chosen, the physical and mechanical issues of camera mounting were solved. Then a software solution was designed using an object-oriented approach after modeling the solution in UML (Unified Modeling Language). The signals from the environment were captured by the vehicle's vision systems and pre-processed . A histogram was used to determine a brightness threshold to determine if a pothole is within the field of view. Then, a binary image is formed. Regions are then detected in the binary image. Regions that have a diameter close to 2 feet and a ratio of circumference to diameter close to 2 feet are considered potholes. (Jai – what do you mean? The ratio of 2Pi * r/2r = Pi) The logic controller where navigational strategies are evaluated uses these signals to decide a final course of navigation.

The primary significance of the solution is that it is interfaced seamlessly into the existing central logic controller. The solution can also be easily extended to detect and avoid any two dimensional shape. (Oh really!) Experiments were conducted with what results?

Acknowledgements

This work depends heavily on the existing design and systems in BEARCAT II and I am greatly indebted to all the students who have worked in this incredible machine. My special thanks to the team of 2000 – you were one great bunch.

Dr. Ernie Hall has been the key force behind the robotics team all these years and has been the primary motivator for my work. My deepest gratitude to him for having supported and inspired me to complete this work.

My thanks are also due to the team of 2001, especially Vidya and Pravin for their help in integrating my work with the existing BEARCAT systems.

I also have to express my gratitude to the Robot Vision class of Spring 2000 for their support and help in the various areas of this research.

My sincere thanks are also due to the committee members for their valuable suggestions. A very special thanks to EPIX Inc for having given us the Imaging Board free of cost.

My love and thanks to Neelu and my parents.

Table of Contents

Use the automatic feature to put a table of contents with page numbers.

Insert page numbers on this list.

List of Figures

Chapter 1

Introduction

Humans have sophisticated vision systems through which a large amount of information can be received and interpreted very quickly. We also have extremely adept systems for taking decisions based on these visual input and also the means to perform appropriate actions. A truly autonomous robot must sense its environment accurately and react appropriately. This issue attains greater importance in an outdoor, variable environment.

The Center for Robotics Research at the University of Cincinnati has been involved in building such an autonomous robot for competing at the International Ground Robotics Contest conducted by the Association for Unmanned Vehicles System International. [01] The contest is divided into three sub events – autonomous challenge, road debris contest and follow-the-leader contest.

Jai – Put your references in a consistent format.Usuall superscripts are used or square brackets but not both.

The autonomous challenge is the main event where the fully autonomous unmanned ground vehicle must negotiate around an outdoor obstacle course under a prescribed time while staying within the 5mph speed limit, and avoiding obstacles in the track. Vehicles must not be controlled by a remote human operator during the competition, and all computational, power, steering and control equipment must be carried onboard.

The obstacle course is normally laid out in grass, pavement or simulated pavement, or any combination, over an area approximately 60 to 80 yards long, by 40 to 60 yards wide. The course boundaries are designated by continuous or dashed white and/or yellow lane markers (lines) approximately three inches wide, painted on the ground. Track width will be approximately 10 feet wide with a turning radius not less then 8 feet.

The obstacle course presents a number of challenges, in the form of artificial inclines, sand pits, 5 gallon white pails, and full-size orange and white construction barrels. Artificial inclines can have a gradient of up to 15%. Sand pits have a sand depth of 2-3 inches, and sometimes may be simulated by a light beige canvas tarp covering the entire width of the track. The placement of white pails and construction barrels is randomized from left, right and center placements prior to every run. A few series of pails/drums simulate obstacles that result in a trap if a wrong turn is chosen by the vehicle.

1.1 Problem Definition

At the competition for this year, a new obstacle was introduced in the navigation course – simulated potholes. These are white circles of size 2 feet diameter placed randomly across the course. These potholes are positioned such that there is always a minimum passage width maintained throughout the navigation course.

The primary objective of this research is to arm the robot with the functionalities required to successfully detect and avoid any simulated potholes in its course.

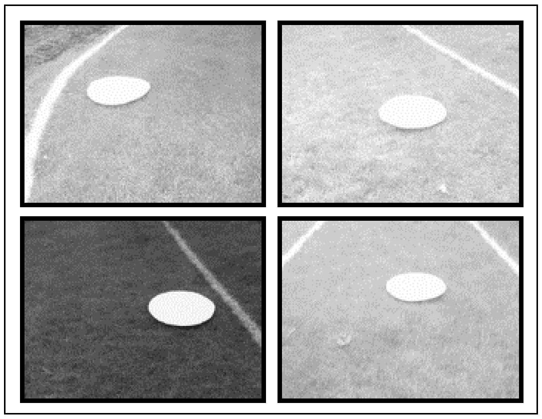


Figure *1* Simulated potholes in the obstacle course

A simulation of the pothole in the course is shown in *Figure 1*.

Number the Figures sequentially and type the word figure. Make sure that you have referred to each figure in the text.

These are only some of the possible scenarios that can occur and they largely depend on the orientation of the robot with respect to the course.

A list of key assumptions taken into consideration while developing the solution are listed below:

- Only circular potholes will be detected. As the camera is mounted in an incline to the horizontal, the circular pothole generates an elliptical image in the camera.
- The size of the pothole must be approximately 2 feet in diameter for them to be detected.

- The solution should perform satisfactorily in a wide variety of lighting conditions.

1.2 Existing System Configuration

The development of the Bearcat II is based upon the realization that the design of a complex electro-mechanical system, like an automated guided vehicle, must be accomplished by a decomposition of the design problem into simpler units. This decomposition would be carried on until all units reach the individual component level. These components can then be designed, integrated to form major sub-units and ultimately, on further integration, lead to the entire system. The major subsystems in the Bearcat II robot are vision system, obstacle avoidance system, steering control system, speed control system, safety system and central logic controller.

1.2.1 Vision System

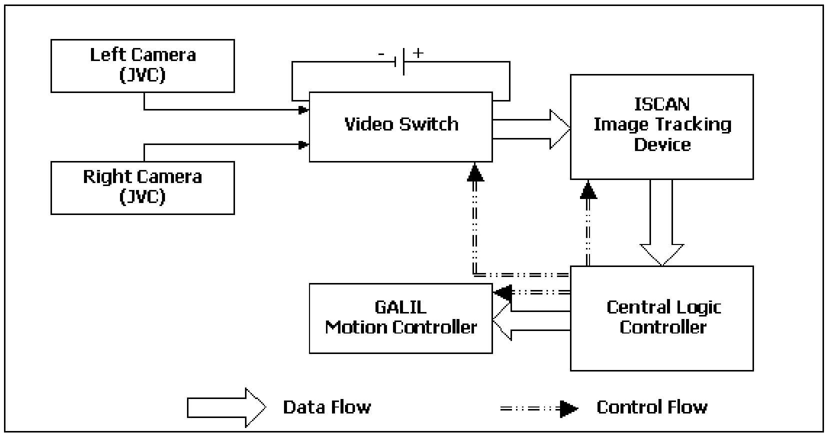


Fig *1-2* Existing Vision System

Two JVC CCD cameras are used to view and follow the left or the right lane marker. Only one lane marker is followed at any instant. A CCSU-8BW video switch device from FSR, Inc. alternates between the two cameras depending on the visibility of the lane marker. The Central Logic Controller controls the video switch through the Galil DMC.

An ISCAN RK-446-R image-tracking device processes the image from the camera. The device finds the centroid of the brightest or darkest region in a computer controlled window, and returns the X, Y co-ordinates of its centroid and the size information of the blob. If no object is found, a loss of track (LOT) signal is generated. The cameras are angled downward at 32° and panned to the front at 30°. This setup gives a 4-foot wide view of the ground and 6-foot view ahead.

Image co-ordinates are two-dimensional while actual world co-ordinates are three-dimensional. In an autonomous situation, the problem is to determine the three-dimensional coordinates of a point on the line given its image coordinates. A new algorithm [04] was developed to establish the mathematical and geometrical relationships between the physical 3-D world coordinates of the line to be followed and its corresponding 2-D digitized image coordinates. The mean square error between these measured and computed points was 0.242 inches for the X-axis and 0.295 inches for the Y-axis, which established the accuracy of the algorithm. A new simpler vision calibration method has also been designed which uses only four 3-D points.

1.2.2 Motion System

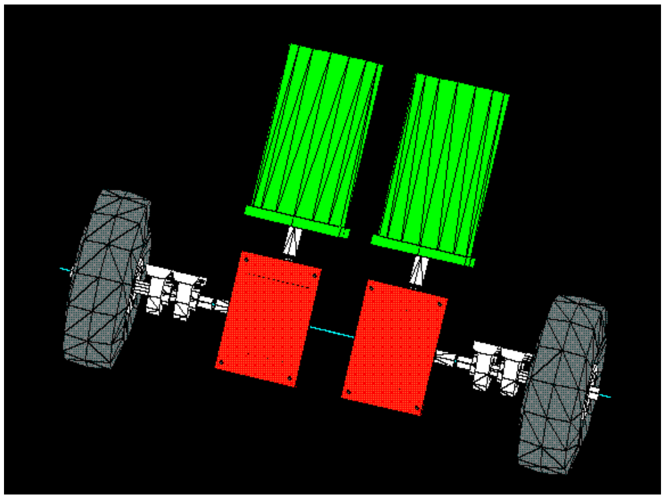


Fig *1-3* Existing Motion System

Two individually driven front wheels powered by two 36-volt, 15-amp motors are used. The motors drive the left and the right wheel separately through two independent gearboxes, which increase the motor torque by a factor of 40. This enables a zero turning radius by rotating one wheel in the forward direction and the other in the reverse direction. This unique design offers the ability to make a turn about the center of axis of the drive wheels thereby providing the vehicle exceptional maneuverability. The power to each motor is delivered from an AMC DC 48A amplifier that amplifies the signal from the Galil DMC Motion Controller. To complete the control loops; a position encoder is mounted on each of the drive motors. A castor wheel at the back of the vehicle balances the load.

Steering the vehicle is achieved by varying the speed of the left and the right wheels while negotiating a curve. This enables the vehicle to make a curved turning path parallel to the track lines. By manipulating the sum and difference of the speed of left and the right wheels, the velocity and the orientation of the vehicle can be controlled at any instant.

Velocity of Vehicle, $$V_M = (V_L + V_R)/2 \tag{1-1}$$

Orientation of Vehicle, $$q = (V_L - V_R)/WT \tag{1-2}$$

where, VL = Velocity of the left wheel

VR = Velocity of the right wheel

W = Distance between the center of the two wheels

T = Sampling time

The design objective in selecting the motor control parameters was to obtain a stable control over the steering system with a good phase and gain margin and a fast unit step response. For this purpose a Galil Motion Control Board with a Proportional Integral Derivative controller (PID Controller) was used. The system was modeled in MATLAB using SIMULINK and the three parameters of the PID controller were selected using a simulation model to obtain optimum response. The unit step response values for the PID controller were tested on the actual vehicle and were fine tuned using the software supplied by Galil Motion, Inc. – WSDK 1000.

1.2.3 Obstacle Avoidance System

Generally motion-planning algorithms for a mobile robot are developed based on the assumption that there are no changes in the surrounding environment. One of the goals in robotics is to endow robots with the ability to move and operate autonomously in an environment with unknown, perhaps moving obstacles. Robot navigation is described as the guiding of a mobile robot to a desired destination or along a desired path in an environment characterized by a terrain and a set of distinct objects, such as obstacles and landmarks. The robot must successfully navigate around obstacles, reach its goal and do so efficiently. Not only must a robot avoid colliding with an obstacle such as a rock, it must also avoid falling into a pit or ravine and avoid travel on terrain that would cause it to tip over.

The purpose of a path planner is to compute a path, i.e., a continuous sequence of configurations. The primary concern of path planning is to compute collision-free paths. Another key issue is the uncertainty problem. It is related to various sources of uncertainty affecting the actual robot (control and sensing errors, inaccurate models of the environment, etc.). Approaches to path planning for mobile robots can be broadly classified into two categories - those that use exact representations of the world and those that use discretized representations. The main advantage of discretization is that adjusting the cell size can control the computational complexity of path planning. In contrast, the computational complexity of exact methods is a function of the number of obstacles and/or the number of obstacle facets, which we cannot normally control.

Chapter 2

Literature Survey

Detection of potholes is an interesting issue and has great significance in the context of highway maintenance. A major problem in planning highway maintenance is the unreliability of manual measurements [02, 03, 04], which can lead to poor prioritization of remedial work. Also, an automated pothole sensing method will result in the generation of huge amount of data that can be used as input into highway maintenance packages. [04, 05] These packages help in the streamlining of maintenance planning. The chief benefit with automated data collection is the surveying of highway conditions could be extended, highway utilization increased and maintenance costs reduced. This is important in both developing and developed countries.

2.1 Review of past approaches

Jai – Be consistent with capitalization in your titles.

Although the detection and measurement of potholes in an automated way has huge benefits, monetary and technical, little has been published about automated sensing methods. Have you done a patent search. If the highway department has these machines, someone has probabally patened them.

Grace, et al. [06] describe an algorithm for using machine vision to estimate the 3D profile of potholes in highways. Structured illumination is used to generate a low-resolution edge map, which is then used to initialize active contour maps to synthesize the final image. The original image is sub-sampled to form two lower resolution intensity images. In the lowest resolution image layer two candidate edge points are generated, by convolving the image with a matched filter that is the expected width of the light stripe. A number of responses equal to the number of projected edge points are selected as candidate edge points. An optimal edge contour is constructed for each stripe in the second image layer and propagated to the first image layer. In the first layer, the position of each edge point is refined by searching a limited spatial domain, and constrained by internal and external energy functions. The set of optimum snakes is propagated to the next higher resolution layer and the process is repeated. The experimental setup described in the paper uses two synchronized Pulnix TM765 cameras, fitted with Tamron 28-70 mm zoom lenses. The images are digitized using a VideoPix frame grabber and are analyzed off-line. From the experimental results depicted in the paper, pothole sizes can be estimated with reasonable accuracy and the irregular 3D shape can be reliably reconstructed. The main demerit of the presented technique is that it is an offline technique and so real-time application may be impractical, if not impossible.

Georgopoulus, et al. [07] discuss an algorithm based evaluation procedure to estimate pavement distress using digital image processing. The algorithm described performs the tasks of recognition of the crack type, estimation of the severity and the calculation of its area. The method proposed is based on the representation of the cracking by one or more vectors, depending on its type, or, in other words, the determination of the X, Y co-ordinates of the cracking on the image. This method is shown to consist essentially of two basic steps: *vectorization* and *image representation*. The first step uses the images produced by the image-processing phase as the input. In this step, the distress is represented by a set of vectors that approximate the cracks composing the distress. A geometrical model of the crack is then obtained which describes the shape, by including the elements of interest. In the second step the computer is taught to distinguish between the various types of cracking. This is the actual decision making step and comprises of three tasks: *cracking classification*, *severity estimation* and *extent evaluation*.

Daehie Hong, et al. [08] present a tethered mobile robot for automating highway operations that uses a laser range finder based sensor. How well did it work?

2.2 Background for Proposed Technique

2.2.1 Histogram

The image intensities $f(i,j)$ in an image can be considered as being random variables with probability density function (pdf) $p_f(f)$. The image pdf carries valuable global information about the image content. However, the pdf is generally not available and must be estimated from the image itself by using the empirical pdf usually called the *histogram*. Assuming our image i has a gray scale level of 256, and that n_k, k = 0...L-1, is the number of pixels having intensity *k*, the histogram $\hat{p}_f(f)$ is given by the relation:

$$\hat{p}(f_k) = \frac{n_k}{n}, k = 0,1 \ldots L-1 \qquad (2\text{-}1)$$

where *n* is the total number of image pixels. [09]

Based on this, a simple algorithm can be developed for constructing the histogram for a gray scale image. The algorithm used in the solution is illustrated in the Figure number below:

Thresholding is one of the important techniques for image segmentation based on the similarity of brightness of image objects. [10] A number of methods have been proposed to calculate the threshold value for an image. [10, 11] The threshold selection techniques can be divided into two groups such as bilevel and multilevel.

In bilevel thresholding, one threshold value is used for segmenting an image into a background and an object. Bilevel thresholding is used if an image has an object distinct from the background. In an image, if the object is distinct from the background, then the histogram of the gray level will be bimodal, and a threshold can be chosen to coincide with the valley of the gray level histogram. Each pixel that has a gray value above the threshold value will be assigned to the background (object) and each pixel with gray value below or equal to the threshold will be assigned to the object.

Multilevel thresholding is used when an image has several objects that are distinct from the background. The presence of several objects makes the histogram multimodal. The threshold is found by locating the valley that separates the objects. Thus the multilevel method is an extension of the bi-level thresholding method. If the gray-level image is neither multimodal nor bimodal, then it may be unimodal. A unimodal histogram has only one peak and thus there is no valley between the two groups of objects. In such cases, where valley seeking thresholding method cannot be used, other methods like criterion functions must be used. [12]

There are instances where images in gray level values, has several objects that are different from the background. In these cases, the histograms do not yield a bimodal plot but rather a complex multi-modal distribution. There are many effective algorithms to solve this problem. [13, 14] But in the simulation studies, it was found that for the given problem, the histograms are always clearly bimodal.

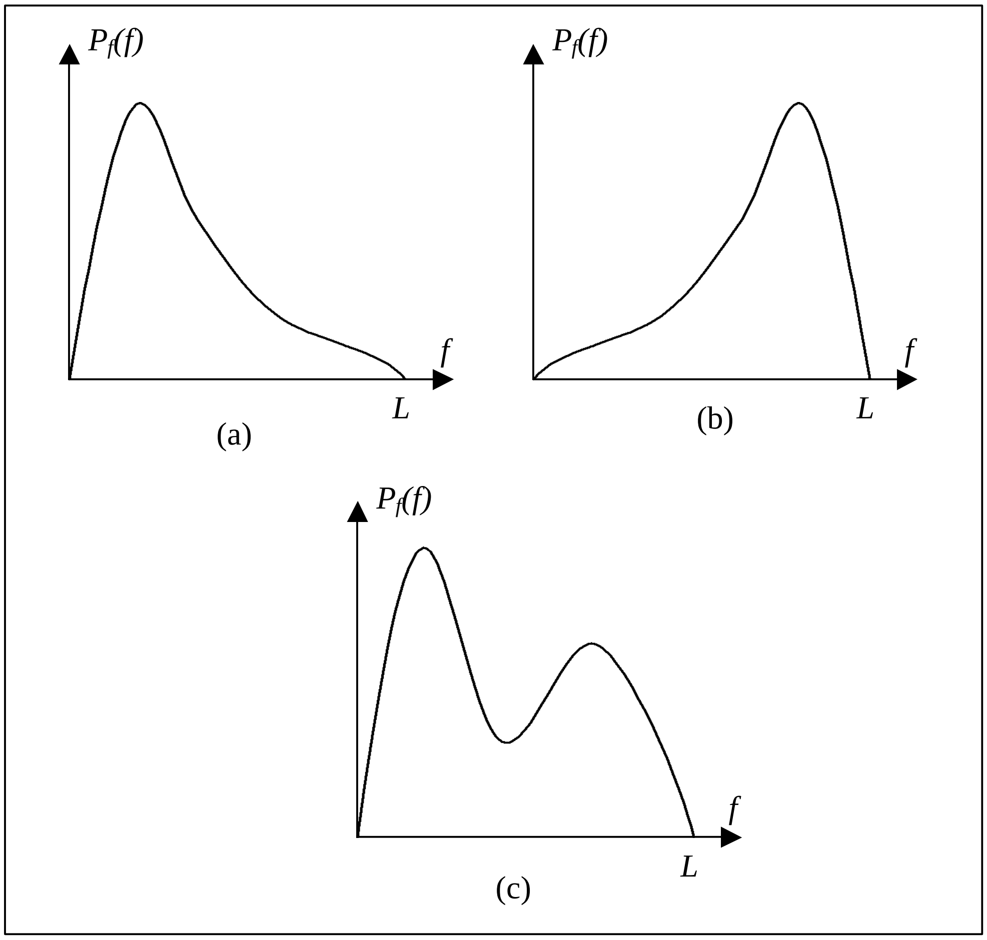


Fig *2-1* Image Histograms

If this is not an original drawing you must acknowledge the source of it.

The image histogram carries important information about the image content. If its pixel values are concentrated in the low image intensities, as can be seen in Fig *2-1*, the image is dark. A bright image has a histogram that is concentrated in the high image intensities as seen in Fig *2-1*(b). The histogram of Fig *2-1*(c) reveals that the image contains two objects with different intensities (or, possibly, one object clearly distinguished from its background). If the image histogram is concentrated on a small intensity region, the image contrast is poor and the subjective image quality is low. Modifying its histogram can enhance image quality.

Simulation trials were conducted for the thresholding operation based on empirical data obtained from a sample navigation course. The results from the simulation trials are shown in the Fig *2-2* below:

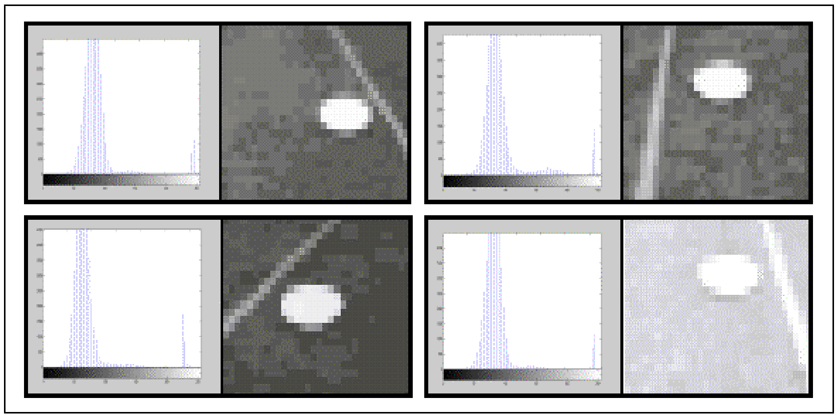


Fig *2-2* Histograms of pothole

The results proved that histogram always followed a bimodal plot, irrespective of the location of the pothole or the orientation of the robot.

From the Fig *2-2* shown above, it can be easily concluded that the first mode depicts the background and the second mode identifies the pothole. Although the lines, representing the navigation course boundary, tend to add noise to the image received by the camera, the degree of this noise is insignificant compared to the total number of pixels in the pothole and the background.

Hence, the second maximum is used to differentiate the pothole from the background. As a result of the histogram operation, the gray scale image is converted into a binary image.

2.2.2 Edge Detection

Boundaries of objects tend to show up as intensity discontinuities in an image. Experiments with the human visual system show that boundaries in an image are extremely important; often an image can be recognized from only a crude outline. [09, 15] This fact provides the principal motivation for representing objects by their boundaries. Also, the boundary representation is easy to integrate into a large variety of object recognition algorithms.

One might expect that algorithms could be designed that finds the boundaries directly from the gray-level values in the image. But when the boundaries have complicated shapes, this is difficult. First transforming the image into an intermediate image of local gray-level discontinuities, or edges, and then composing these into a more elaborate boundary have obtained much greater success. This strategy reflects the principle: when the gap between the representations becomes too large, introduce intermediate representations. In this case, boundaries that are highly model-dependant may be decomposed into a series of local edges that are highly model-independent.

A local edge is a small area in the image where the local gray levels are changing rapidly in a simple way. An *edge operator* is a mathematical operator (or its computational equivalent) with a small spatial extent designed to detect the presence of a local edge in the image function.

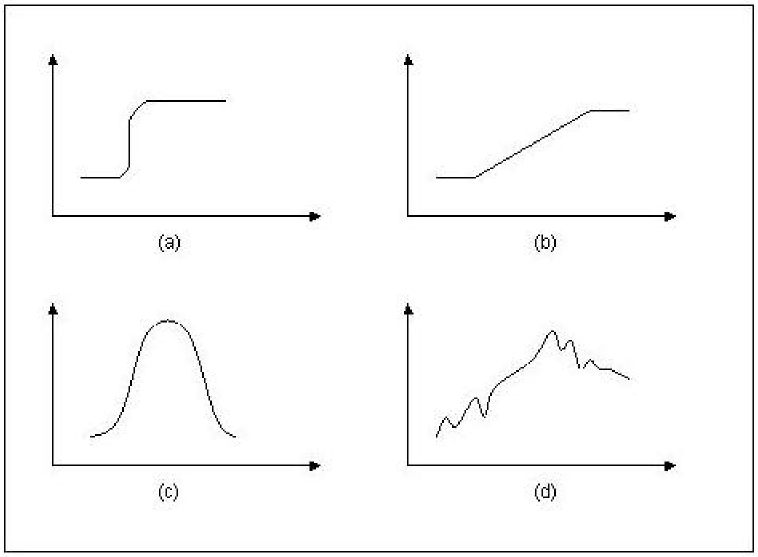


Fig *2-3* Different Edge Profiles

It was found that specifying a priori which local edges correspond to relevant boundaries was a little difficult. Depending on the particular task domain, different local changes will be regarded as likely edges. Plots of gray level versus the distance along the direction perpendicular to the edge for some hypothetical edges [16] are shown in Fig *2-3*. This figure demonstrates some different kinds of *edge profiles* that are commonly encountered. Of course, in most practical cases, the edge is noisy (Fig *2-3*(d)) and may appear as a composite of profile types. The fact that different kinds of edge operators perform best in different task domains has prompted the development of a variety of operators. However, the unifying feature of most useful edge operators is that they compute a *direction* that is aligned with the direction of the maximum gray-level change, and a *magnitude* describing the severity of the change. Since edges are a high-spatial-frequency phenomenon, edge finders are also usually sensitive to high-frequency noise, such as ‘snow’ on a television screen or film grain.

Operator fall into three main classes: (1) operators that approximate the mathematical gradient operator, (2) template matching operators that use multiple templates at different orientations, and (3) operators that fit local intensities with parametric edge models.

2.2.2.1 Types of edge operators

Gradient and Laplacian

The most common and historically earliest edge operator is the gradient. [17] For an image function $f(\mathbf{x})$, the gradient magnitude $s(\mathbf{x})$ and direction $f(\mathbf{x})$ can be computed as

$$s(\mathbf{x}) = (\Delta_1^2 + \Delta_2^2)^{1/2} \qquad (2\text{-}2)$$
$$f(\mathbf{x}) = \tan^{-1}(\Delta_2 / \Delta_1) \qquad (2\text{-}3)$$

where

$$\Delta_1 = f(x+n, y) - f(x, y) \qquad (2\text{-}4)$$
$$\Delta_2 = f(x, y+n) - f(x, y) \qquad (2\text{-}5)$$

n is a small integer, usually unity, and atan (x, y) and tan^{-1} (*x/y*) must be in the proper quadrant. The parameter n is called the 'span' of the gradient. Roughly, n should be small enough so that the gradient is a good approximation to the local changes in the image function, yet large enough to overcome the effect of small variations in f .

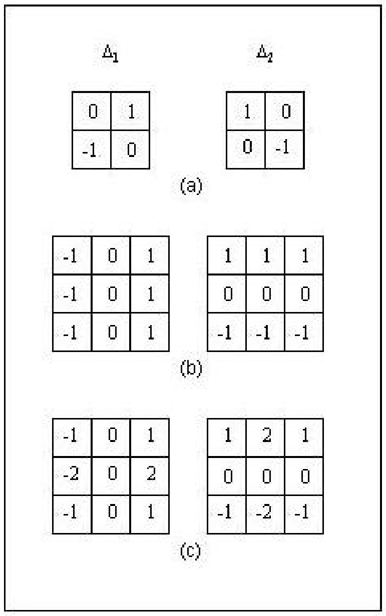


Fig *2-4* Gradient Difference Operators

Same for this figure.

Equation *2-4* is only one *difference operator*, or way of measuring gray level intensities along orthogonal directions using Δ_1 and Δ_2. Fig *2-4* shows the gradient difference operators compared to other operators. [18] The reason for the modified operators of Prewitt and Sobel is that local averaging tends to reduce the effects of a noise. These operators do, in fact, perform better than the Robert's operator for a step edge model.

The *Laplacian* is an edge detection operator that is approximation to the mathematical Laplacian $\partial^2 f / \partial x^2 + \partial^2 f / \partial y^2$ in the same way that the gradient is an approximation to the first partial derivatives. One version to the discrete Laplacian is given by: [16]

$$L(x,y) = f(x,y) - \tfrac{1}{4}\left[f(x,y+1) + f(x,y-1) + f(x+1,y) + f(x-1,y) \right] \qquad \textit{(2-6)}$$

The Laplacian has two disadvantages as an edge measure:

1. useful directional information is not available, and
2. the Laplacian, being an approximation to the second derivative, double enhances any noise in the image.

Because of these disadvantages, the Laplacian edge detector was not used for this work.

Edge Templates

The *Kirsch operator* [19] is related to the edge gradient and is given by

$$S(\mathbf{x}) = \max\left[1, \max_k \sum_{k-1}^{k+1} \left| f(\mathbf{x}_k) - f(\mathbf{x}) \right| \right] \qquad \textit{(2-7)}$$

where $f(\mathbf{x}_k)$ are the eight neighboring pixels to $\mathbf{x}$ and where subscripts are computed modulo 8. A 3-bit direction can also be extracted from the value of k that yields the maximum in Equation *2-7*. Four separate templates are matched with the image and the operator reports the magnitude and direction associated with the maximum match.

2.2.2.2 Boundary Detection

The goal of boundary detection is to make a coherent one-dimensional (*edge*) feature from many individual local edge elements. There are a number of methods to detect boundaries and the selection of any one method is highly dependent on the implicit or explicit constraints on a given grouping. If there is high knowledge about the image that is processed, then the global form of the boundary and its relation to other image structures is highly constrained. On the other hand, little prior knowledge means that the segmentation must proceed more on the basis of local clues and evidence in general (domain-dependent) assumptions with fewer expectations and constraints on the final resulting boundary.

The different methods considered for boundary detection in this work are:

1. *The Hough Transform*. This technique comes most handy when detecting boundaries whose shape can be described in an analytical or tabular form.
2. *Graph Searching*. This method represents the image of edge elements as a graph. Thus a boundary is a path through the graph.

3. *Contour Following*. This is identical to the graph searching technique and works well when there is a higher level of knowledge about the image data beforehand.

The Hough Method for Curve Detection

The classical Hough technique for curve detection is applicable if little is known about the location of the boundary, but its shape can be described as a parametric curve (e.g., a straight line or a conic). The main advantages in using the Hough transform is that it is relatively unaffected by gaps in curves and by noise.

To introduce the Hough technique, [20] consider the problem of detecting straight lines in images. Assume by some process image points have been selected that have a high likelihood of being on linear boundaries. The Hough technique organized these points into straight lines, basically by considering all possible straight lines at once and rating each on how well it explains the data.

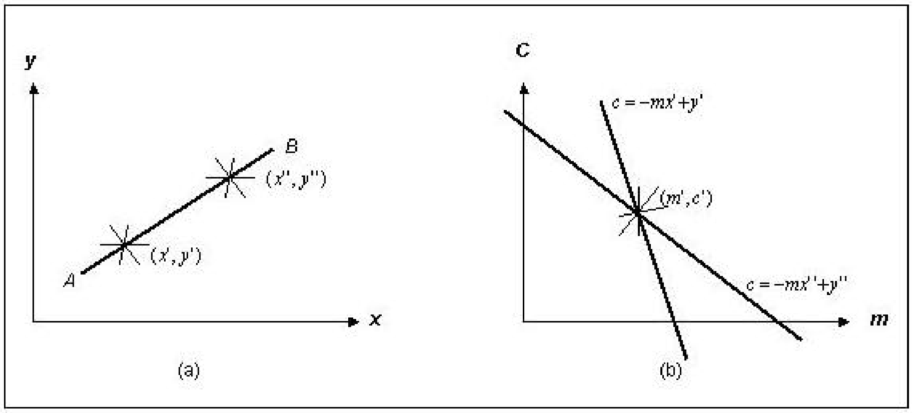
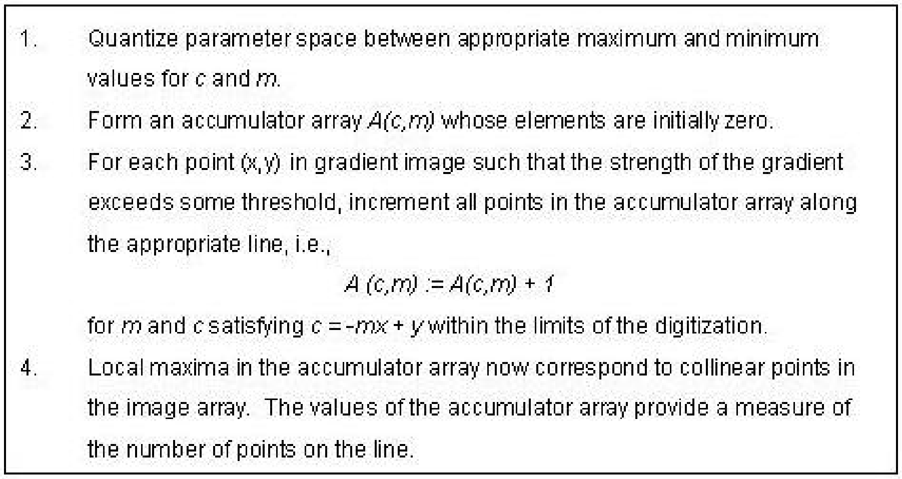


Fig *2-5* Hough Technique

Same here.

Consider the point $\mathbf{x}'$ in Fig *2-5*, and the equation for a line, $y = mx + c$. Now, the possible lines that could pass through $\mathbf{x}'$ are those with m and c satisfying $y' = mx' + c$. Regarding (x', y') as fixed, the last equation is that of a line in *m*-*c* space, or parameter space. Repeating this reasoning, a second point (x'', y'') will also have an associated line in parameter space and, further more, these lines will intersect at the point (m', c'), which corresponds to the line *AB* connecting these points. In fact, all points on the line *AB* will yield lines in the parameter space which intersect at the point (m', c') as shown in Fig *2-5*.

This relation between image space $\mathbf{x}$ and parameter space suggests the following algorithm for detecting lines:



1. Quantize parameter space between appropriate maximum and minimum values for *c* and *m*.
2. Form an accumulator array *A(c,m)* whose elements are initially zero.
3. For each point (x,y) in gradient image such that the strength of the gradient exceeds some threshold, increment all points in the accumulator array along the appropriate line, i.e.,

   A (c,m) := A(c,m) + 1

   for *m* and *c* satisfying *c = -mx + y* within the limits of the digitization.
4. Local maxima in the accumulator array now correspond to collinear points in the image array. The values of the accumulator array provide a measure of the number of points on the line.

Fig *2-6* Algorithm for Hough Technique

This technique is generally known as Hough technique. [21]

Since *m* may be infinite in the slope-intercept equation, a better parametrization of the line is $x\cos q + y\sin q = r$. This produces a sinusoidal curve in (r,q) space for fixed *x*, *y*, but otherwise the procedure is unchanged.

The generalization of this technique to other curves is straightforward and this method works for any curve $f(\mathbf{x},\mathbf{a}) = 0$, where "**a**" is a parameter vector. In the case of a circle parametrized by

$$(x-a)^2 + (y-b)^2 = r^2 \tag{2-8}$$

for fixed **x**, the modified algorithm for *(2-8)* increments values of *a*, *b*, *r* lying on the surface of a cone. Unfortunately, the computation and size of the accumulator array increases exponentially as the number of parameters, making this technique practical only for curves with a small number of parameters.

The Hough method is an efficient implementation of a generalized matched filtering strategy. For instance, in the case of a circle, imagine a template composed of a circle of 1's (at a fixed radius *r*) and 0's everywhere else. If this template is convolved with the gradient image, the result is the portion of the accumulator array $A(a,b,r)$.

In its usual form, the technique yields a ser of parameters for a curve that best explains the data. The parameters may specify an infinite curve (e.g., a line of parabola). Thus, if a finite curve segment is desired, some further processing is necessary to establish end points.

Graph Searching

A graph is a general object that consists of a set of nodes $\{n_i\}$ and arcs between nodes $\langle n_i, n_j \rangle$. A numerical weight or *cost* can be associated with each of the arcs. The search for the boundary of an object is cast as a search for the lowest-cost path between two nodes of a weighted graph.

Assume that a gradient operator is applied to the gray-level image, creating the magnitude image $S(\mathbf{x})$ and the direction image $f(\mathbf{x})$. Now interpret the elements of the direction of the image $f(\mathbf{x})$ as nodes in a graph, each with a weighting factor $S(\mathbf{x})$. Nodes $\mathbf{x}_i$, $\mathbf{x}_j$ have arcs between them if the contour directions $f(\mathbf{x}_i), f(\mathbf{x}_j)$ are appropriately aligned with the arc directed in the same sense as the contour direction.

Figure xx.u shows the interpretation. To generate xx.y, impose the following restrictions. For an arc to connect from $\mathbf{x}_i$ to $\mathbf{x}_j$, $\mathbf{x}_j$ must be one of the three possible eight-neighbors in front of the contour direction $f(\mathbf{x}_i)$ and, furthermore, $S(\mathbf{x}_i) > T, S(\mathbf{x}_j) > T$, where T is a chosen constant, and $\left|\left\{\left[f(\mathbf{x}_i) - f(\mathbf{x}_j)\right] \bmod 2p\right\}\right| < p/2$.

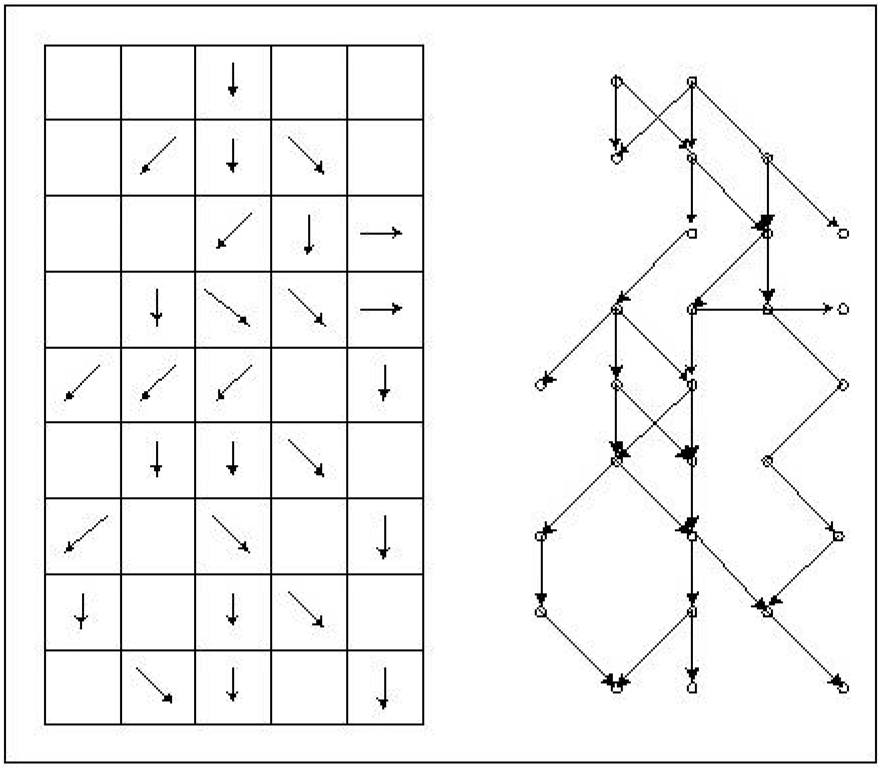


Fig *2-7* Graph Searching

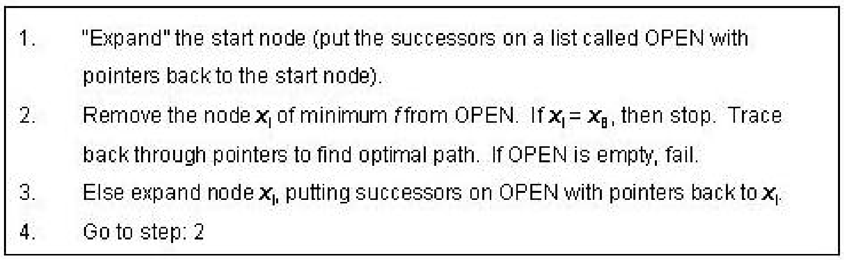
To generate a path in a graph from $\mathbf{x}_A$ to $\mathbf{x}_B$ one can apply the well-known technique of heuristic search. [22] The specific use of heuristic search is to follow edges in images was first proposed by Martelli. [23] Suppose:

1. That the path should follow contours that are directed from $\mathbf{x}_A$ to $\mathbf{x}_B$
2. That we have a method for generating the successor nodes of a given node (such as the heuristic described above)

3. That we have an evaluation function $f(\mathbf{x}_j)$ which is an estimate of the optimal cost path from $\mathbf{x}_A$ to $\mathbf{x}_B$ constrainted to go through $\mathbf{x}_j$

Nillson expresses $f(\mathbf{x}_j)$ as the sum of two components: $S(\mathbf{x}_i)$, the estimated cost of journeys from the *start node* $\mathbf{x}_A$ to $\mathbf{x}_i$, and $h(\mathbf{x}_i)$, the estimated cost of the path from $\mathbf{x}_i$ to $\mathbf{x}_B$, the *goal node*.

With these preliminaries, Nillson states the heuristic algorithm as,



1. "Expand" the start node (put the successors on a list called OPEN with pointers back to the start node).
2. Remove the node $\mathbf{x}_i$ of minimum f from OPEN. If $\mathbf{x}_i = \mathbf{x}_B$, then stop. Trace back through pointers to find optimal path. If OPEN is empty, fail.
3. Else expand node $\mathbf{x}_i$, putting successors on OPEN with pointers back to $\mathbf{x}_i$.
4. Go to step: 2

Fig *2-8* Heuristic for Graph Searching

The component $h(\mathbf{x}_i)$ plays an important role in the performance of the algorithm; if $h(\mathbf{x}_i) = 0$ for all *I*, the algorithm is a *minimum-cost search* as opposed to a *heuristic search*. If $h(\mathbf{x}_i) > h^*(\mathbf{x}_i)$ (the actual optimal cost), the algorithm may run faster, but may miss the minimum cost path.

Contour Following

If nothing is known about the boundary shape, but regions have been found in the image, the boundary can be recovered by a simple edge-following operation: “blob finding” in images. Given a binary image, the goal is to find the boundaries of all distinct regions in the image.

This can be done by a simple procedure that performs like Papert’s turle: [24]

1. Scan the image until a region pixel is encountered.
2. If it is a pixel region, turn left and step; else, turn right and step.
3. Terminate upon return to the starting pixel.

This procedure requires the region to be four-connected for a consistent boundary. Parts of n eight-connected region can be missed.

A slightly more elaborate algorithm described by Rosenfield [25], generates the boundary pixels exactly. The algorithm works by first finding a four-connected background pixel from a known boundary pixel. The next boundary pixel is the first pixel encountered when the eight neighbors are examined in a counter clockwise order from the background pixel.

Once the appropriate threshold is set, the edges in the image can be detected. The edges are detected by comparing the values of the neighborhood pixels. Four edge detection operators were compared for the given problem using MathCAD and the Robert’s Edge Detection Operator was selected.

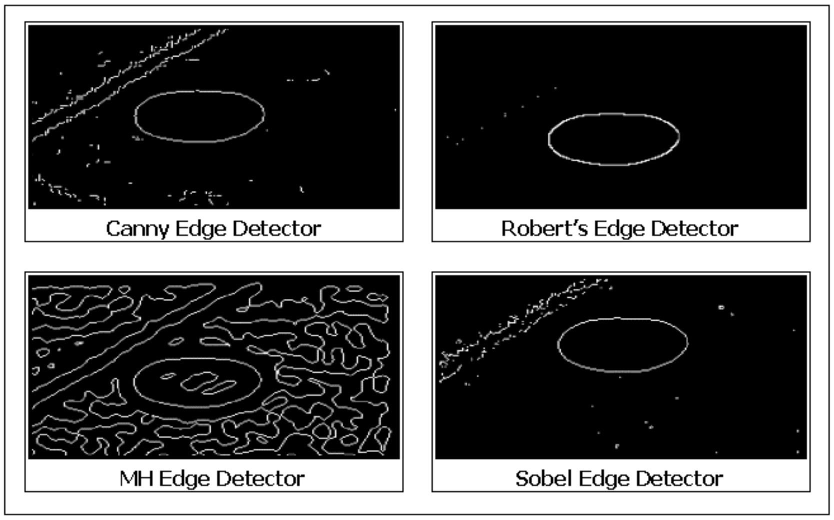


Fig *2-9* Comparison of Edge Operators

Fig *2-9* presented above illustrates the performance of different edge detection operators for one particular scenario of the pothole image. The comparisons were also performed for a number of different scenarios and in all the cases, the Robert's Edge Detection Operator performed better than the other operators.

2.3.3 Blob Analysis

Ang, et al. [26] present blob analysis as a tool for retrieving artifact images using multidimensional, multi-resolution features. This paper describes a method of image retrieval based on multidimensional shape and blob features. In addition, feature analysis is applied to multi-resolution images to extract features details at different perceptual scales. Each feature set forms a 4-D feature vector in a multidimensional multi-resolution feature space. Image decomposition with the wavelet transform is used for the extraction of blobs and for multi-resolution feature analysis.

Image content is described in terms of object shape and painted design. Object shape is characterized by measurements made from the object boundary: compactness, contour moment, aspect ratio, and region concavity. The regions with painted patterns are characterized by total blob area, blob count, blob dispersion and blob moment.

The patterned regions, which contain high-frequency detail, are extracted by thresholding the high frequency sub bands in the wavelet domain. This is followed by region-filtering procedure to remove “noisy” blobs (small isolated regions), and a merging procedure to merge perceptually significant larger neighboring blobs into larger ones. The following measures are used in this paper to abstract and characterize the object shape and texture blobs:

Total blob area. The sum of the blob areas A_T is normalized by the object area A_O, to give the normalized feature:

$$A_b = \frac{A_T}{A_O}. \qquad (2\text{-}9)$$

A high value of A_b is an indication that a large part of the object is covered with a painted design.

Blob count. The number of distinct textured regions or blobs in a picture N_b is a useful parameter for describing the perceptual "busyness" of image detail. A low value indicates a relatively simple painted design, while a high value indicates a very detailed design.

Blob dispersion. The structural information of image details may be described by the spatial arrangement of texture blobs. This is measured by their distribution or dispersion with respect to the object center:

$$D_b = \frac{1}{N_b \overline{R}} \sum_{i=1}^{N} d_i \qquad \textit{(2-10)}$$

where d_i is the distance between the centroids of the *i*th blob and the object and $\overline{R}$ is the average radial distance of the object boundary. If all the blobs are concentrically arranged, this parameter has zero value. A larger value of D_b indicates that the image details in the object are dispersed, presenting a relatively complex design.

Blob moment. The blob moment measures the overall distribution of the mass or skewness of the blobs with respect to the object center:

$$M_b = \frac{1}{A_T \overline{R}} \sum_{i=1}^{A_T} p_i \ , \qquad \textit{(2-11)}$$

where p_i is the distance between a pixel in a blob and the centroid of the object.

Chapter 3

Solution Architecture

The schematic of the solution for detection of simulated potholes is illustrated in Fig *3-1*.

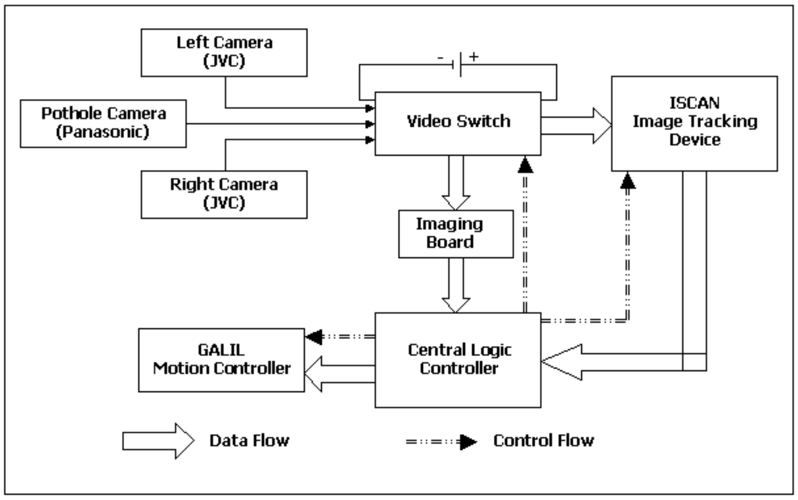


Fig *3-10* Solution Schematic

As painting white circles on the course simulates the potholes, the vision system is a reasonable solution to detect the potholes. At the same time, the existing two cameras cannot be used for this purpose as they were tied up in the line following system. A monochrome Panasonic CCD camera is used to capture the course ahead of the robot. A monochrome camera was selected because then the data would be compact compares to a color camera. This would reduce the computational complexity of the system.

The data from the camera is fed into the video switch and the central logic controller is used to select a single camera for data processing.

The image data from the Panasonic camera is directly passed to the central computer through an imaging board. The reasons for using an Imaging Board as an intermediary between the central logic controller and the camera are:

1. The imaging board reduces the computational overheads required for complex imaging calculations at the central logic controller.
2. The image manipulation routines used by the imaging board are compact and efficient.
3. Off-the-shelf library functions are available for image manipulation, which can reduce the solution development time significantly.

3.1 Design of Physical Parameters

The location and orientation of the camera plays a vital role in the correct functioning of the pothole detection system. The orientation parameters determine the view of the camera and consequently the size and shape of the resulting image of the simulated pothole. The camera should be located at the highest point in the front side of the robot in order to have the largest possible field-of-view. The task of locating the camera was made easy by the modular construction of the robot. The fastening method and the clamps used are detailed in section 4.1 Camera Installation.

One of the important considerations during the design of the physical parameters for the camera is the reaction distance [27] of the robot. This distance is the minimum distance within which an obstacle needs to be detected for the robot to successfully avoid it.

The reaction distance for the Bearcat II robot was taken as 12 feet. This means that the pothole must be detected at a distance no less than 12 feet for the robot to successfully maneuver away from it. The present design of the robot allows the camera to be placed at a maximum height of 4 feet. Based on these physical parameters, the angle at which the camera needs to be oriented is calculated to be 71°.

$$q = \tan^{-1}\left(\frac{\text{Reaction Distance}}{\text{Height of camera}}\right) = \tan^{-1}\left(\frac{12}{4}\right) = 71^{o} \qquad \textit{(3-1)}$$

With this geometry, a pothole in the center of the track is approximately ? pixels wide.

3.2 Selection of Imaging Board

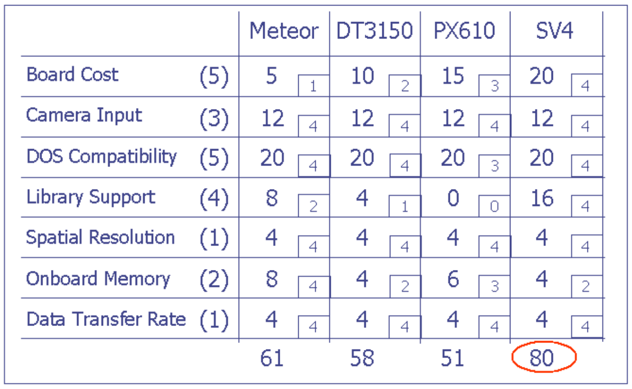
A number of constraints needed to be born in mind before a suitable imaging board could be selected for the application. These are:

Operating System: The robot is built over the DOS operating system and most of the hardware in the robot use the DOS interrupt vectors for communication with the Central Logic Controller. Hence the imaging board needs to be compatible with the DOS operating system.

PCI Host Interface: The existing hardware on the robot use all the available ISA and USB ports in the computer motherboard. So the imaging board needed to be interfaced through a PCI Host Interface.

Monochrome Input: As the data comes from a monochrome CCD camera, the imaging board had to be monochrome-compatible.

Based on these initial constraints, four Imaging Boards were selected – Meteor of Matrox, Inc., DT3150 of Data Translation, PX610 of Imagenation and PIXCI SV4 of EPIX, Inc. A decision matrix was then developed to select one among these four Imaging Boards based on chosen selection criteria. The decision matrix that was developed is illustrated in Fig *3-2*.



		Meteor		DT3150		PX610		SV4	
Board Cost	(5)	5	1	10	2	15	3	20	4
Camera Input	(3)	12	4	12	4	12	4	12	4
DOS Compatibility	(5)	20	4	20	4	20	3	20	4
Library Support	(4)	8	2	4	1	0	0	16	4
Spatial Resolution	(1)	4	4	4	4	4	4	4	4
Onboard Memory	(2)	8	4	4	2	6	3	4	2
Data Transfer Rate	(1)	4	4	4	4	4	4	4	4
		61		58		51		80	

Fig *3-2* Decision Matrix for Imaging Board Selection

The decision matrix lists the four products that have to be compared in the top row, and the various parameters on which the decision has to be based in the first column. The various parameters considered in this decision matrix are cost of the imaging board, input received from the camera, compatibility to DOS, presence of imaging function libraries, spatial resolution, onboard memory and data transfer rate.

Each of these parameters is also associated with a weight factor, which is shown alongside. This weight factor for each parameter depends on the context in which the decision matrix is built. A higher value indicates that the parameter is more important and a lower value indicates that the parameter is less important. Once the weight factor is decided for each parameter, each of the different products is compared and a corresponding cost factor is assigned for each parameter. In this case, SV4 Imaging Board from EPIX is cheaper than the other three so a cost factor was assigned to it. On the other hand, Meteor from Matrox has the lowest cost factor of 1, as it is the costliest of the four imaging boards.

Jai – don't use random upper case. Save it for proper nouns and anachroms.

The cost factor is multiplied with the weight factor of each parameter and the sum of the result is calculated to give the overall significance factor for each imaging board. The significance factor for each imaging board is listed in the last row.

Based on the decision matrix, the PIXCI SV4 Imaging Board manufactured by EPIX, Inc., was selected for the solution.

3.3 Solution Methodology

The methodology for detecting the presence of a pothole consists of a series of imaging operations performed on the image obtained from the camera. All these operations are performed at the imaging board to reduce the computational load on the central computer that controls the robot.

As it is with any image processing solution, the raw image from the camera needs to be pre-process before any operations can be performed over it. The data from the camera is in the form of an array of pixel values, with each of the pixel represented in a gray scale value ranging between 0 and 255. This image is captured by the imaging board and stored in a frame buffer. All the operations are performed on this image buffer and the resulting data about the presence of a pothole is sent to the central logic controller.

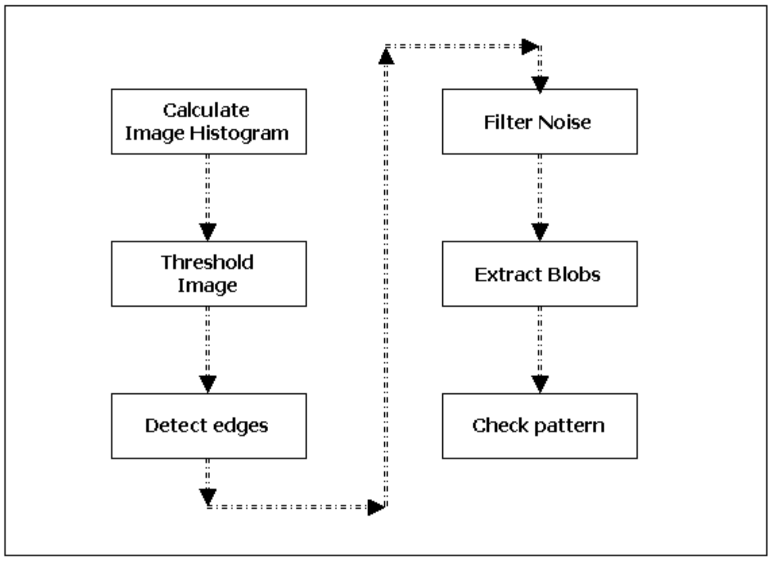


Fig *3-3* Solution Methodology

Is this really correct? Are you using the detected edges in your solution?

The series of imaging operations performed on the image buffer is explained in the control flow diagram shown in Fig *3-3*:

3.4 Software Design

Software architecture deals with the design and implementation of the high-level structure of the software. According to Phillipe Kruchien [28], it is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability.

3.4.1 Object Oriented Framework

A model composed of multiple *views* or perspectives needs to be used to describe software architecture. In this research, the software is described through two views – the *logical* view and the *process* view.

The *logical* view is the object model of the design and primarily supports the functional requirements – what the system should provide in terms of services to its end users. The system is decomposed into a set of key abstractions, taken from the problem domain, in the form of *objects* or *object classes*. They exploit the principles of abstraction, encapsulation and inheritance. The decomposition is not only for the sake of functional analysis, but also serves to identify common mechanisms and design elements across various parts of the system.

The *process* view captures the concurrency and synchronization aspects of the design. The process architecture takes into account some non-functional requirements, such as performance and availability. It addresses the issues of concurrency and distribution, of fault tolerance, and most importantly, how the main abstractions from the logical view fit within the process architecture – on which thread of control is an operation for an object actually executed. A *process* is basically a grouping of tasks that form an executable unit.

The software is partitioned into a set of independent *tasks*. A task is defined as a separate thread of control. The next section describes Unified Modeling Language, which is used to generate the software models used in this research. The *process* and *logical* views of the software model that have been generated in this research are explained in section 3.3.3.

3.4.2 Unified Modeling Language

The Unified Modeling Language (UML) is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system. [29]

The UML is appropriate for modeling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems. It is a very expressive language, addressing all the views needed to develop and then deploy such systems. Even though it is a very expressive, the UML is not difficult to understand and to use. Learning to apply the UML effectively starts with forming a conceptual model of the language, which requires three major elements: the UML's basic building blocks, the rules that dictate how these building blocks may be put together, and some common mechanisms that apply throughout the language.

The UML is a language that provides a vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modeling language is one whose vocabulary and rules focus on the conceptual and physical representation of a system.

Modeling yields an understanding of a system. No one model is ever sufficient. Rather, you often need multiple models that are connected to one another in order to understand anything but the most trivial system. The vocabulary and rules of a language such as the UML tell you how to create and read well-formed models.

3.4.2.1 Benefits of using UML

The UML is a language that can be used for visualizing, specifying, constructing, documenting the artifacts of a software system.

UML as a language for visualizing

For many programmers, the distance between thinking of an implementation and then pounding it out in code is close to zero. You think it, you code it. In fact, some things are best cast directly in code. Text is a wonderfully minimal and direct way to write expressions and algorithms.

In such cases, the programmer is still doing some modeling, albeit entirely mentally. One may even sketch a few ideas in a paper. However, there are several problems with this. First, communicating those conceptual models to others is error-prone unless everyone involved speaks the same language. Typically, projects and organizations develop their own language, and it is difficult to understand what's going on if you are an outsider or new to group. Second, there are some things about a software system that you can't understand unless you build models that transcend the textual programming language. For example, the meaning of a class hierarchy can be inferred, but not directly gasped, by studying the system's code. Third, if the developer who cut the code never wrote down the models that are in his or her head, this information would be lost forever or, at best, only partly re-creatable from the implementation, once the developer moved on.

UML is a language for specifying

In this context, specifying means building models that are precise, unambiguous, and complete. In particular, the UML addresses the specification of all the important analysis, design, and implementation decisions that must be made in developing and deploying a software system.

UML is a language for constructing

Although UML is not a graphical programming language, its models can be directly connected to a variety of programming languages. This means that it is possible to map from a model in the UML to a programming language such as C++ or Java, or even to tables in a database, or the persistent store of an object-oriented database.

This mapping permits forward engineering: the generation of code from a UML model into a programming language. The reverse is also possible: you can reconstruct a model from the implementation back into the UML. Combining these two paths of forward code generation and reverse engineering yields round-trip engineering, meaning the ability to work in either a graphical or a textual view, while tools keep the two views consistent.

UML is a language for documenting

The UML model in itself is like a detailed documentation of the system under development. Additionally, various UML notations can be used to convey meanings and interpretations to the base model.

3.4.2.2 A Conceptual Model of UML

The vocabulary of the UML encompasses three kinds of building blocks:

1. Things
2. Relationships

3. Diagrams

1. Things

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

There are four kinds of things in the UML:

1. Structural things
2. Behavioral things
3. Grouping things
4. Annotational things

Of these, the two important things that are relevant to this research are the structural and grouping things.

Structural things: Structural things are the nouns of UML models. These are mostly the static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

A *class* is a description of a set of objects that share the same attributes, relationships and semantics. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations as in the figure.

An *interface* is a collection of operations that specify a service of a class or a component. An interface therefore describes the externally visible behavior of that element.

A *collaboration* defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements.

A *use case* is a description of set of sequence of actions that a system performs that yield an observable result of value to a particular actor.

The other three elements - active class, component, and node - are not relevant to our research.

Grouping things: Grouping things are the organizational part of UML models. There are the boxes into which a model can be decomposed. In all there is only one kind of grouping thing, namely, packages.

A *package* is a general-purpose mechanism for organizing elements into groups. Structural things, behavioral things and even other grouping things may be placed in a package. Graphically, a package is rendered as a tabbed folder, usually including its name and its contents, as in figure.

Annotational Things: Annotational things are the explanatory parts of the UML models. These are the comments that one applies to describe, illuminate and remark about any element in a model. There is again one primary kind of an annotational thing, called a note.

A *note* is simply a symbol for rendering constraints and comments attached to an element or a collection of elements. Graphically a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment.

2. Relationships

There are four kinds of relationships in the UML:

1. Dependency
2. Association
3. Generalization
4. Realization

Dependency: A dependency is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing). Graphically, a dependency is rendered as a dashed line, possibly directed.

Association: An association is a structural relationship that describes a set of links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, an association is rendered as a solid line, possibly directed and often containing other adornments like multiplicity and role names.

Generalization: A generalization is a specialization/generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent). In this way, the child shares the structure and the behavior of the parent. Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.

Realization: A realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. A realization relationship is typically encountered in two places: between interfaces and components that realize them, and between use cases and collaborations that realize them. Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship.

3. Diagrams

A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). Diagrams are useful in visualizing a system from different perspectives, so a diagram is a projection into a system. Most often a diagram represents an elided view of the different elements in a system, and so the same elements might appear in all diagrams, in a few diagrams or none. There are nine types of diagrams in UML:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Statechart diagram
7. Activity diagram
8. Component diagram

9. Deployment diagram

A *class diagram* shows a set of classes, interfaces and other collaborations and their relationships. Class diagrams address the static design view of a system.

An *object diagram* shows a set of objects and their relationships. Object diagrams represent static snapshots of instances of the things that are found in class diagrams.

A *use case diagram* shows a set of use cases and actors (a special kind of class) and their relationships. They address the static use case view of a system.

An *interaction diagram* shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. A sequence diagram is an interaction diagram that emphasizes the time ordering of messages; a collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.

A *statechart diagram* shows a state machine, consisting of states, transitions, events and activities. Statechart diagrams address the dynamic view of a system and are especially important in modeling the behavior of an interface or class and emphasizing the event-ordered behavior of an object.

An *activity diagram* is a special kind of statechart diagram that shows the flow from activity to activity within a system. They are especially important in modeling the functions of a system and emphasize the flow of control among objects.

A *component diagram* shows the organizations and dependencies among a set of components.

A *deployment diagram* shows the configuration of run-time processing nodes and the components that live on them. They address the static deployment view of architecture.

3.4.3 Software Model

The model is basically algorithmic oriented with all the sub-systems designed in an object-oriented manner to promote software reusability and management. The whole software was designed as a stand-alone package that can be compiled and linked to the existing control software. The software is implemented in a number of classes and hence data is encapsulated from the main software. The user only needs to create an instance of the `PotResult` class and can get the information about the presence of a pothole by accessing the respective member functions.

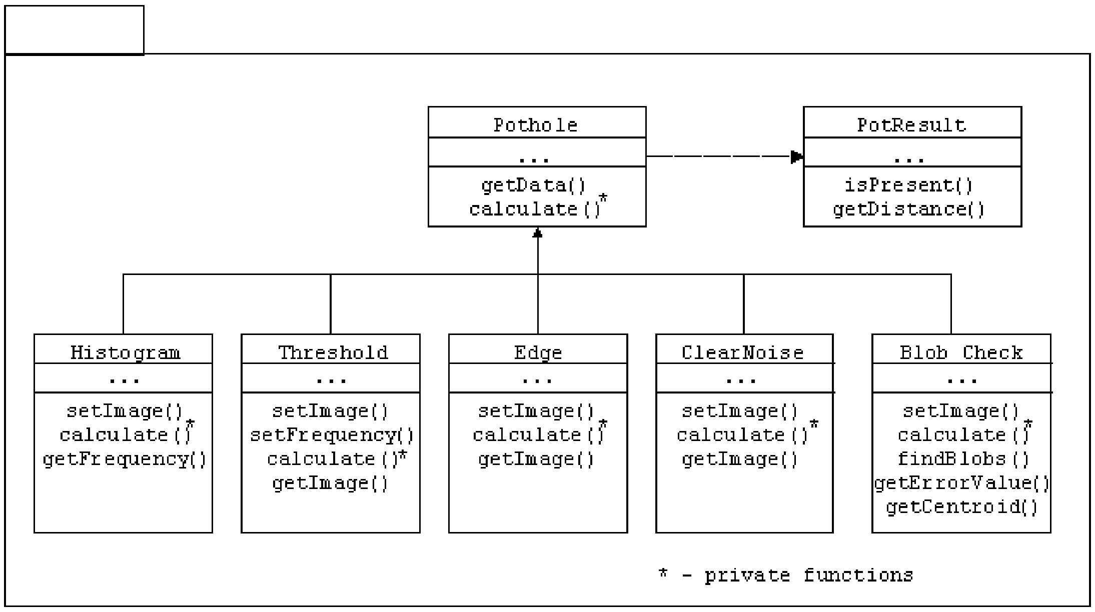


Fig *3-4* Software Model

The model illustrated in Fig *3-4* details the construction of the software. Each of the boxes describes a class within the model. The classes in the second tier are the derived classes from the `Pothole` class, and are executed in the order in which they are listed below. All the classes have private data members that hold the image buffer in a 256x256 integer array.

The raw image supplied by the frame grabber is a 256x256 array of pixel values with each of the pixels holding a gray-scale value. A histogram is first developed for the image buffer and a threshold operation is performed to convert the gray scale image into a binary image. An edge detection operation is then performed. At this stage a simulated pothole, if present, would be in the shape of an oblong ellipsoid. There is also a possibility for noise to be present in the image due to various sources – like the lines marking the course or even the obstacles. A noise removal operation is performed to remove noises from the image. A pattern matching logic is used to identify noises from lines and other obstacles.

A blob analysis is performed on the image to identify and extract all the blobs within the image buffer. Based on the physical parameters of the camera, an error range has been established for the potential image sizes of these blobs. Each of the blobs identified within the image frame is first checked for the ellipsoid pattern, and the blobs that match this pattern are then filtered using the permissible error values calculated from the physical parameters of the camera.

If there is a blob that falls within the specified error range, then the `PotResult` object is set to a Boolean value of true and the centroid of the blob is also calculated and passed to the `PotResult` object.

Chapter 4

Implementation on BEARCAT

4.1 Camera Calibration Procedure

The Panasonic monochrome camera has to be calibrated before it can be used for the detection of potholes. The calibration of the camera primarily consists of setting up the focal length of the camera and the aperture size of the lens. Since the Panasonic camera used in this research was a manual camera, this setup has to be done by hand before actual use of camera.

How do you set the focal length? To what value? How do you set the aperture? To what value for morning sunshine? What value for evening? What about night?

The EPIX imaging board comes with a imaging software called XCIP which can be used for this calibration purpose. The following procedure has to be used to configure and use XCIP in order to calibrate the camera.

EPIX Imaging board will not work with an extended memory setup. Hence, before using the XCIP software, any extended memory setup must be removed from the configuration files of the operating system. With the memory set to “low” the following steps have to be completed to calibrate the camera.

- Go to c:\XCIP folder
- Key XCIPLITE and press Enter

- Choose Setup --> Video Format & Resolution --> Digitize Video Format
- Choose !set Format : RS170 and press Enter
- Press Quit Menu once to come back to the Video Format & Resolution menu
- Choose Video Capture & Display
- Change the value of !Video Input Multiplexer to 1
- Change selection from :Digitize to :Capture

At this point, the image captured by the pothole camera is displayed on the right top end of the screen. Adjust the angle of the camera if needed to suit the "reaction distance" of the robot. Adjust the focal length and the aperture of the camera so that the image is clear and sharp.

Can you put a screen capture of what this looks like?

4.2 Integration of Software

The design and development of the software in this research has been conceived carefully so that the integration of the software with the existing BEARCAT II software will be seamless.

As mentioned in Section 3.4.3, the software model for the proposed solution is modular. The integration of this software to the existing software is achieved by creating an instance of the `PotResult` object and calling the public functions of this object – namely the `isPresent` function and the `getDistance` function.

The `isPresent` function returns a Boolean variable that indicates the presence of a pothole in the image captured by the Panasonic monochrome camera.

The getDistance function returns distance of the pothole from the current position of the camera. This dimension of this length variable is “feet”.

Additional information about the pothole – like the size of the pothole and the world co-ordinates – can be obtained by calling the `getResult` function. This function returns a `potResult` object that has member functions for returning the x and y co-ordinates in the world co-ordinate system.

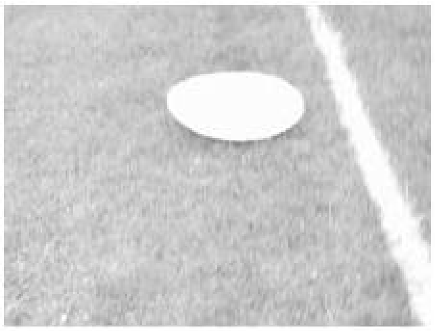
Jai – I don’t think your result ever got integrated into the g-cart program. Let’s do it. This will take some effort but will let you answer the question about your experimental results.

Chapter 5

Results and Analysis

The pothole software solution was tested under various conditions both inside the lab and at the competition course during the International Ground Robotics Contest 2000 at Orlando, Florida. Different factors were changed during the course of testing like lighting conditions, course track (grass, pavement) and camera settings.

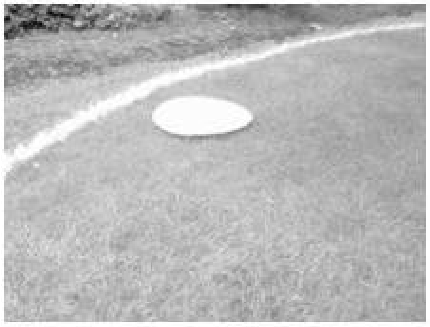
The results from the testing are listed below. The image observed by the Panasonic monochrome camera is listed on the left side and the output generated by the pothole detection solution is listed in the right side for each instance of the test.



Results Reported

isPresent	true
getDistance	2.2 feet
Co-ordinates	(0.1, 2.2)

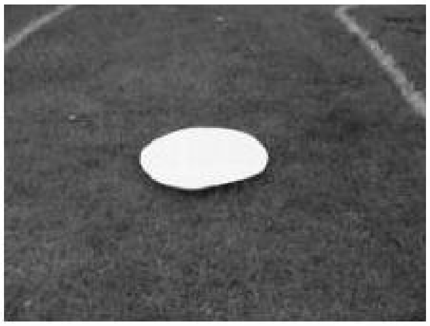
Fig *5-1* Pothole to the right of the robot



Results Reported

isPresent	true
getDistance	2.9 feet
Co-ordinates	(-0.4, 2.9)

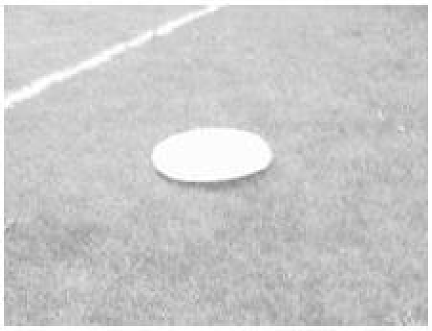
Fig *5-2* Pothole in a curve in the course



Results Reported

isPresent	true
getDistance	2.2 feet
Co-ordinates	(0.1, 2.2)

Fig *5-3* Pothole in the center of the course



Results Reported

isPresent	true
getDistance	1.8 feet
Co-ordinates	(0.1, 1.8)

Fig *5-4* Pothole to the left of the robot

Shouldn't there be some comparison of hand measured values with image values. I don't believe this has been done.

Chapter 6

Conclusions and Recommendations

This research has been an excellent learning experience in various engineering disciplines. The solution for detection and avoidance of potholes has been successfully integrated into the existing BEARCAT II system. Most importantly this integration has been made seamlessly without affecting the performance of any other part of the system.

The software for the research has been done in a modular, object-oriented fashion leading to a design that is highly reliable and easy to maintain. The software has been organized in both logical oriented and process oriented approaches. This process could be extended to the entire software operating the BEARCAT II machine. Generation of such an object-oriented framework for the BEARCAT II software will lead to a less steeper learning curve for new members working on the robot. Also this framework would ease the addition of new systems to the BEARCAT more straightforward.

The blob selection technique used in this research is very straightforward in the sense that the algorithm tries to find blob matches based on the dimensional similarity to a pothole. This algorithm could be further refined and heuristic based approach could be tried. Also, other blob characteristics like blob moment, shape features could also be used to detect the presence of potholes in the environment.

References

01 AUVSI, http://www.secs.oakland.edu/SECS_prof_orgs/PROF_AUVSI/

02 S. A. Guralnick, E. S. Suen and C. Smith, *Automatic inspection of highway pavement sufaces*, Journal of Transport Engineering, Vol. 119, No. 1, pp. 1-12, 1993.

03 R. J. Hintz, C. Karakadas and J. Kang, *Analysis of pavement cracking and rutting using close range photography*, Engineering Remote Sensing, Vol. 55, No. 2, pp. 217-221, 1989.

04 M. S. Snaith, *The development and implementation of pavement management systems: a case study over 15 years*, Process Instrumentation in Civil Engineering Transportation, Vol. 129, pp.72-90, 1996.

05 T. Watanatala, C. G. Harral, W. D. Patterson, A. M. Dhareshwar, A. Bandari and K. Tusnokawa, *The highway design and maintenance standards model*, Johns Hopkins University Press, Baltimore, MD, 1987.pages

06 A. E. Grace, D. Pycock, H. T. Tillotson and M. S. Snaith, *Active shape from stereo for highway inspection*, Machine Vision and Applications, Vol. 12, pp. 7-15, 2000.

07 A. Georgopoulus, A. Loizos and A. Flouda, *Digital image processing as a tool for pavement distress evaluation*, International Journal of Photogrammetry and Remote Sensing, Vol. 50, No. 1, pp. 23-33, 1995.

08 Daehie Hong, Steven A. Velinsky and Kazuo Yamazaki, *Tethered mobile robot for automating highway maintenance operations*, Robotics and Computer Integrated Manufacuring, Vol. 13, No. 4, pp. 297-307, 1997.

09 Ernie L. Hall, *Digital image processing algorithms and applications*, John Wiley & Sons, Inc., 2000.

10 R. C. Gonzalez and R. E. Woods, *Digital image processing*, Addison-Wesley Reading, MA, 1992.pages

11 A. Jain, *Fundamentals of digital image processing*, Prentice-Hall, Engelwood Cliffs, NJ, 1989.pages

12 Prasanna Sahoo, Carrye Wilkins and Jerry Yeager, *Threshold selection using renye's entropy*, Pattern Recognition, Vol. 30, No. 1, pp. 71-84, 1997.

13 J. N. Kapur, P. K. Sahoo, and A. K. C. Wong, *A new method for gray level picture thresholding using the entropy of the histogram*, Computing Vision Graphics Image Process, Vol. 29, pp. 273-285, 1985.

14 F.J.Chang, J.C.Yen and S.Chang, *A new criterion for automatic multilevel thresholding*, IEEE Transaction Image Process, Vol. 4, pp. 370-378, 1995.

15 F. Attneave, Some informational aspects of visual perception, Pschycological Review, Vol. 61, 1954.pages

16 Dana H. Ballard and Christopher M. Brown, *Computer vision*, Prentice-Hall, Engelwood Cliffs, NJ, 1982.pages

17 L. G. Roberts, *Machine perception of three-dimensional solids*, Optical and Electro-optical Information Processing, MIT Press, Cambridge, MA, 1965.pages

18 J. M. S. Prewitt, *Object enhancement and extraction*, Picture Processing and Pshycopictorics, Academic Press, New York, 1970.pages

19 R. A. Kirsch, *Computer determination of the constituent structure of biological images*, Computers and Biomedical Research, Vol. 4, No. 3, pp. 315-328, 1971.

20 R. O. Duda and P. E. Hart, *Use of the hough transformation to detect lines and curves in pictures*, Commun. ACM, Vol. 15, No. 1, pp. 11-15, 1972.

21 P. V. C. Hough, *Method and means for recognizing complex patterns*, U.S. Patent 3,069,654, 1962.

22 N. J. Nilsson, *Problem solving methods in artificial intelligence*, McGraw-Hill, New York, 1971.pages

23 A. Martelli, *Edge detection using heuristic search methods*, Computer Graphics and Image Processing, Vol. 1, No. 2, pp. 169-182, 1972.

24 R. O. Duda and P. E. Hart, *Pattern recognition and scene analysis*, Wiley, New York, 1973.pages

25 A. Rosenfield, *Picture processing by computer*, Academic Press, New Tork, 1968.pages

26 Y. H. Ang, S. H. Ong, and Zhao Li, *Retrieval of artifact images using multidimensional multiresolution features*, Computer Graphics, Vol. 20, No. 1, pp. 51-59, 1996.

27 Ramesh Thyagarajan, *A motion control algorithm for steering the AGV in an outdoor environment*, MS Thesis, University of Cincinnati, Ohio, 2000.

28 Phillipe Kruchien, *Architectural blueprints – the "4+1" view model of software architecture*, IEEE Software, Vol. 12, No. 6, pp. 42-50, 1995.

29 Grady Booch, James Rumbaugh and Ivar Jacobson, *The unified modeling language user guide*, Addison-Wesley, Massachusetts, 1999.pages

Appendix A

Software Source Code

```
#define FORMAT "RS-170"


/*
 *      Set 'define' parameters below according to PC
 *   configuration, such as desired image memory
 *   size. If NULL, the default parameters are used.
 *   Alternately, this could be modified to use
 *   getenv("PIXCI"), GetPrivateProfileString(...),
 *      RegQueryValueEx(...), or any other convention
 *   chosen by the programmer.
 *
 */
#define CONFIG "-IM 4096 -QP 0x2D"


/*
 *  NECESSARY INCLUDES:
 */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <stdarg.h>
#include "pxipl.h"
#include "pxextras.h"
#include "xcobj.h"
#include "pxobj.h"

#include "pot.h"       //Function Prototypes

struct pximage image;
struct pxip8histab histogram;
struct pxip8histstat stat;
struct pxip8histperc statpercentile;
//struct pxyd location;
struct pxip8blob location;
int highbound, lowbound;
ulong pixcount;


/*
 *  SUPPORT STUFF:
 *
 *  Catch CTRL+C and floating point exceptions so that once
 *  opened, the PIXCI(R) library is always closed before exit.
 *  In some environments, this isn't critical; the operating
 *  system advises the PIXCI(R) driver that the program has
 *  terminated. In other environments, the operating system
 *  isn't as helpful and the driver would be left open.
 */
void signaled(int sig)
{
  if(sig == SIGINT)
    printf("Break\n");
  if(sig ==SIGFPE)
    printf("Float\n");
}


/*
 *  SUPPORT STUFF:
 *
 *  Slow down execution speed so
 *  each step/function can be observed.
*/
void user(char * mesg)
{
  if(mesg && *mesg)
     printf("%s\n", mesg);
  printf("\nContinue (press ENTER) ");
  while(fgetchar() != '\n') ;
  printf("\n");
}



/*
 * FUNCTION:    pxd_xcopen(format, parms)
 *
 * Open the XCOBJ C Library for use.
 */
int pot_open(void)
{
  int i;

  if(pxd_chkstack(1) < 0)
     return(PXERROR);
  i = pxd_xcopen(FORMAT, CONFIG);
  if(i >= 0)
    // user("Open OK");
    printf("");
  else {
     printf("Open Error\n");
     pxd_chkfault(1);
     user("");
  }
  return(1);
}



/*
 * FUNCTION:    pxd_imsize()
 *
 * Report image memory size
 */
void pot_imsize(void)
{
  printf("Image Memory Size: %lu KB\n", pxd_imsize());
  user("");
}



/*
 * FUNCTION:    pxd_vidsize()
 *
 * Select image resolution, for example to reduce the image
 * buffer size, requiring less PCI bandwidth (such as on
 * early PCI machines which don't support 132 Megabytes/sec).
 */
void pot_vidsize(void)
{
  printf("Image Resolution before pxd_vidsize\n");
  printf("xdim       = %d\n", pxd_xdim());
  printf("ydim       = %d\n", pxd_ydim());

  //pxd_vidsize(128,640,-1,96,240,-1,0);

  printf("Image Resolution after pxd_vidsize\n");
  printf("xdim       = %d\n", pxd_xdim());
  printf("ydim       = %d\n", pxd_ydim());
  printf("colors     = %d\n", pxd_cdim());
  printf("bits/pixel = %d\n", pxd_bdim());
  user("");
}



/*
 * FUNCTION:    pxd_video(mode, buffer)
 *
 * Acquire and capture images.
 */
void pot_video1(void)
{
  struct pxfault potFault;
  pxd_video('z', 1L);
  if(pxdrv_fault(&potFault)) {
     printf("Error ");
     printf("%s %l %l\n", potFault.mesg, potFault.arg1,
potFault.arg2);
  }
  //user("Continuously digitizing into buffer 1");
}



/*
 * FUNCTION:    pxd_video(mode, buffer)
 *
 * Acquire and capture images.
 */
void pot_video2(void)
{
  struct pxfault potFault;
  pxd_video('p', 0L);
  if(pxdrv_fault(&potFault)) {
     printf("Error ");
     printf("%s %l %l\n", potFault.mesg, potFault.arg1,
potFault.arg2);
  }
  //user("Image captured into buffer 1");
}


/*
 * FUNCTIONS:    pxd_iopen()
 *         pxd_bdim()
 *         pxd_ios()
 *
 * Display the digitized stream of the image in the form
 * of a two dimensional array for a specific window on the
 * image defined by the following two parameters.
 */
#define potXDIM    640
#define potYDIM    240

void pot_display(void)
{
  int i,x,y;
  uchar   monoimage_buf8[potXDIM*potYDIM];
  ushort  monoimage_buf16[potXDIM*potYDIM];


  if( (i=pxd_iopen(0,1L,0,0,potXDIM,potYDIM, 'r')) != 1 ) {
     printf("pxd_iopen error = %s\n", pxerrnomesg(i));
     user("");
     return;
  }
  user("Image I/O Access Opened");

  if(pxd_bdim() <= 8) {
     if( (i=pxd_ioc(0,monoimage_buf8, potXDIM*potYDIM)) !=
potXDIM*potYDIM) {
        printf("pxd_ioc error: %d != %d\n", i, potXDIM*potYDIM);
        user("");
        return;
     }
  }
  else {
        if( (i=pxd_ios(0,monoimage_buf16, potXDIM*potYDIM)) !=
potXDIM*potYDIM) {
            printf("pxd_ios error: %d != %d\n", i,
potXDIM*potYDIM);
            user("");
            return;
        }
  }

  user("Image Area of Interest Transfered");

  for(y=0; y<potYDIM; y++) {
     printf("I = ");
     for(x=0;x<potXDIM;x++)
        if( pxd_bdim() <= 8)
           printf("%4d ", monoimage_buf8[y*potXDIM+x] );
        else
           printf("%4d ", monoimage_buf16[y*potXDIM+x] );
     printf("\n");
  }
  user("");
}



/*
 * FUNCTION:    pxip8_histab()
 *
 *
 * Display the digitized stream of the image in the form
 * of a two dimensional array for a specific window on the
 * image defined by the following two parameters.
 */
void pot_hist(void)
{
  int potx, poty;

  potx = pxd_xdim();
  poty = pxd_ydim()>>pxd_ylace();
  if( pxvid_setpximage( &image, PXSDIGI, 1L) == PXERROR)
     printf("pximage not set properly\n");

  if( pxip8_histab( NULL, &image, &histogram) == PXERROR)
     printf("histab not completed\n");
}



/*
 * FUNCTION:    none
 *
 *
 * Display the digitized stream of the image in the form
 * of a two dimensional array for a specific window on the
 * image defined by the following two parameters.
 */
void pot_binary(void)
{
   //code to find the two highest pixel-count occurences.
   int i,temp,max,error;
   char *name = "binary.bmp";
   struct pximage imageresult;

   for(i=0;i<(histogram.histpix-1);i++)
     if(histogram.count[i] > max) {
       temp = i;
       max = histogram.count[i];
     }
   histogram.count[temp] = 0;
   max = 0;
   for(i=0;i<(histogram.histpix-1);i++)
     if(histogram.count[i] > max) {
       highbound = i;
       max = histogram.count[i];
     }
   pixcount = histogram.count[highbound];
   histogram.count[highbound] = 0;
   max = 0;
   for(i=0;i<(histogram.histpix-1);i++)
     if(histogram.count[i] > max) {
        lowbound = i;
        max = histogram.count[i];
   }
   /* this function is not used. am using my own logic as above
   if( (pxip8_histstat(&histogram, &stat, &statpercentile)) != 0
)
      printf("Could not calculte histogram statistics\n"); */
   /*if(
(pxip8_pixthreshold2(NULL,&image,&imageresult,lowbound,highbound,
255,0)) == PXERROR )
     printf("Image Contrast Operation failed\n");
   else {
     pxio8_bmpwrite(NULL, &imageresult, NULL, name, 8, NULL);
     printf("Binary image saved\n");
   } */

}



/*
 * FUNCTIONS:    pxip8_bloblist()
 *
 *
 * Display the digitized stream of the image in the form
 * of a two dimensional array for a specific window on the
 * image defined by the following two parameters.
 */
struct pxip8blob pot_blobs(void)
{
   struct pxy        search;
   struct pxip8blob  results[100];
   struct pxip8blob  potresult;
   struct pxywindow  bounds;
   ulong ngood, nbad, bad;
   int r,i,maxarrayindex;
   ulong max;

   maxarrayindex = 0;
   max = 0;
   search.x       = -1;
   search.y       = 0;
   bounds.nw.x    = 0;
   bounds.nw.y    = 0;
   bounds.se.x    = pxd_xdim();
   bounds.se.y    = pxd_ydim()<<pxd_ylace();
   nbad = ngood = 0;

   for(;;) {
     r = pxip8_bloblist( NULL, &image, &search, 'e'^'q' ,
highbound, 0&PXIP8BLOB_CONVEX,
                    &bounds, 0, NULL, 10, results, &bad);
     if(r < 0) {
        printf("Error %s\n", pxerrnomesg(r));
        break;
     }
     nbad += bad;
     ngood += r;
     if( !r )
        break;
     for(i=0; i < r; i++) {
        //printf("Blob at (%f,%f), area=%ld\n",
results[i].ucom.xd, results[i].ucom.yd, results[i].xyarea);
```

```
        if( results[i].xyarea > max ) {
           max = results[i].xyarea;
           maxarrayindex = i;
           location = results[i];
        }
     }
   }
   /*for(i=0;i<r;i++)
     if(results[i].xyarea > max) {
        max = results[i].xyarea;
        maxarrayindex = i;
   } */
   printf("Total blobs: Good=%ld, Bad=%ld\n", ngood, nbad);
   printf("Max blob: %d - sized %ld, at (%f,%f)\n",
maxarrayindex, max, location.ucom.xd, location.ucom.yd);

   //location.xd = results[maxarrayindex].ucom.xd;
   //location.yd = results[maxarrayindex].ucom.yd;
   potresult = location;
   return potresult;
}



/*
 * FUNCTIONS:    pxip8_3x3sobel()
 *
 *
 * Display the digitized stream of the image in the form
 * of a two dimensional array for a specific window on the
 * image defined by the following two parameters.
 */
void pot_edge(void)
{
  int potx, poty;
  char *name = "edged.bmp";

  potx = pxd_xdim();
  poty = pxd_ydim()<<pxd_ylace();

  //sip = pxd_defimage(1L, 0, 0, potx, poty);
  if( pxvid_setpximage( &image, PXSDIGI, 1L) == PXERROR)
     printf("pximage not set properly\n");

  if( pxip8_3x3sobel(NULL, &image, &image, 1) != 0 )
     printf("Error during edge detection\n");
  else {
     pxio8_bmpwrite(NULL, &image, NULL, name, 8, NULL);
     printf("Edged image saved\n");
  }
}
```

```
/*
 * FUNCTION:    pxd_bmpsave()
 *
 * Save image in the .bmp format.
 */
void pot_save(void)
{
  int    i;
  char   *name = "image.bmp";

  printf("Current Image Buffer being saved to file %s\n", name);
  if( (i=pxd_bmpsave(name, 1L, 0,0, -1,-1)) < 0) {
     printf("error in pcd_bmpsave= %s\n", pxerrnomesg(i));
     user("");
     return;
  }
  user("Image Buffer Saved");
}



/*
 * FUNCTION:    pxd_close()
 *
 * Close the PIXCI(R) imaging board
 */
void pot_close(void)
{
  pxd_close();
  //user("PIXCI Imaging Board closed");
}




main(void)
{
  struct pxip8blob result;
  signal(SIGINT, signaled);
  signal(SIGFPE, signaled);

  if(pot_open() < 0)
     return(1);

  //pot_imsize();
  //pot_vidsize();
  pot_video1();
  pot_video2();
//  pot_display();    //currently not used due to stack overflow
  //pot_save();
  pot_hist();
  pot_binary();
//  pot_edge();
  result = pot_blobs();

  printf("\nat (%f,%f)\n", result.ucom.xd, result.ucom.yd);

  pot_close();
  return(0);
}
```

Appendix B

EPIX Architecture and Features

Architecture

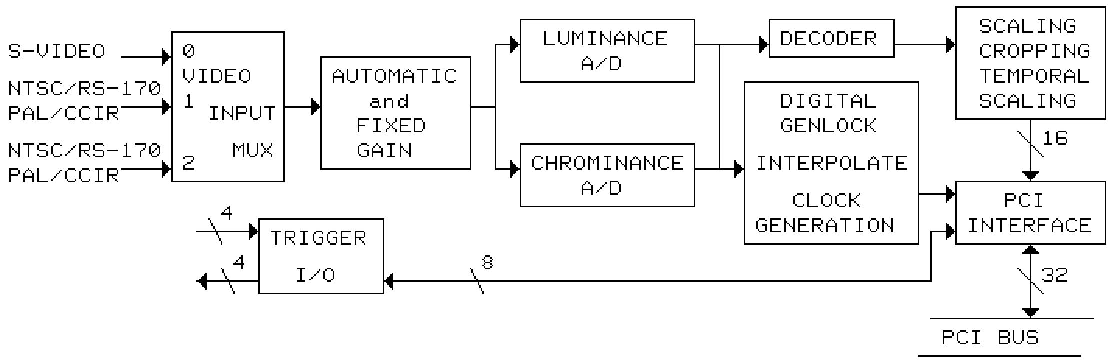


Fig *B-1* Block Diagram of EPIX Imaging Board

Video Input Mux

The multiplexer selects the video source for the Automatic and Fixed Gain from the S-Video input connector (mux 0), from the BNC1 connector (mux 1), or from the BNC2 connector (mux 2). The multiplexer is designed to be switched during vertical blanking.

Automatic and Fixed Gain

The PIXCI® SV4 is configured for Fixed Gain by providing a fixed reference voltage for the A-D converters. Optionally, it can be modified to provide automatic gain for the A-D converters. Automatic gain automatically compensates for reduced amplitude in the analog signal input by adjusting the reference voltage for the A-D converters. Automatic gain, hue, brightness, saturation, and contrast adjustments can be programmed from 0% to more than 200%. The automatic gain control adjusts the video reference level for the A-D converters until the back porch of the Y video input is equal to a programmable value, nominally 0x38. Three other values may be selected for non-standard sync height to video, they are: 0x30, 0x34, and 0x3C.

Luminance A/D

Provides analog to digital conversion of NTSC, RS-170, CCIR, PAL, and the luminance (Y) component of S-Video sources. A programmable prefilter is used for horizontal luminance scaling to reduce aliasing artifacts. The luminance gain can be selected from values of 0% to 236.57%. The luminance and chrominance A-D converters run at 28.636363 MHz for NTSC video formats and 35.46895 MHz for PAL video formats. Digitizing the video signals at twice the typical sampling rate provides low noise digitization. The video inputs only need to be band limited to 14 MHz instead of the 7 MHz limit that would be required at the typical sampling rate. The chrominance and luminance inputs from the S-Video connector and each BNC input connector have low pass filters in the video input. The inputs to the A/D converters are AC coupled.

Chrominance A/D

Provides analog to digital conversion of the color (C) component of S-Video. A chroma comb filter may be enabled or disabled by software. The chrominance U component gain can be selected from values of 0% to 201.18%. The chrominance V component gain can be selected from values of 0% to 283.89%. The hue can be selected from values of -89.3 degrees to +90 degrees.

Decoder

The decoder separates the Y/C components and generates the U/V color difference signals. The PIXCI® SV4 imaging board converts analog video inputs from standard video sources such as S-Video, NTSC, RS-170, PAL, and CCIR to YCrCb in a 4:2:2 format.

Digital Genlock

The digital genlock circuit is used for precise digitization. It interpolates lines with lengths less than or greater than the programmed number of pixels. It generates the pixel clock for transferring image data to the PCI bus interface. The digital genlock circuit automatically recognizes unstable signals (for example, from a VCR) and adapts its locking mechanism to accommodate the source.

Scaling, Cropping

Interpolation is used to scale images down to 1/14 of their original size. Independent horizontal and vertical down scaling of the video input can be selected from ratios of 1:1 to 1:14. For horizontal scaling, 32 phase interpolation is used to accurately determine the value of a pixel. For vertical scaling, an 8 phase interpolation filter is used with a 768 pixel line store.

The window of video to be captured may be cropped in single pixel increments, then scaled in ratios from 1:1 to 1:14, down to as few as 4 pixels by 1 line of image data. Horizontal and vertical scaling is performed in real-time by interpolation, providing an accurate representation of the original image. Either frames or fields of video can be selected.

Some of the possible video formats are:

- 640 by 480 for NTSC square pixels,
- 720 by 480 for NTSC CCIR 601,
- 720 by 576 for PAL CCIR 601, and
- 768 by 576 for PAL square pixels.

Temporal Scaling

Image sequences may be captured at full or reduced frame rates. Temporal decimation can reduce the frame rate down to one image per second.

Trigger I/O

Four input and four output TTL triggers can be used for synchronization with external events. The trigger signals are controlled by the host CPU.

For applications requiring asynchronous reset of the camera, it should be noted that PIXCI® SV4 requires two vertical sync intervals prior to digitizing the video correctly.

PCI Interface

The PIXCI® SV4 imaging board is designed to use the 132 Mbytes/second, burst mode transfer rate of the PCI bus. As a bus master, the PIXCI® SV4 imaging board sends image data to the PCI bus; it does not wait for the computer's CPU to read images from the board into PC memory.

Each PCI bus data cycle can move two pixels (32 bits). If burst mode is not used, the transfer can take several PCI bus cycles. If burst mode is used, two pixels can be transferred on each PCI bus data cycle. When digitizing full resolution video formats and scaling or cropping is not being performed to reduce the number of pixels to be transferred to the PCI bus, burst mode is required to avoid loosing pixels. The PCI bus can only achieve 132 MB/S data transfer rates by using burst mode. If burst mode is not supported or is disabled; only scaled, cropped, or reduced resolution images can be transferred on the PCI bus.

Image sequences may be captured at full or reduced frame rates, onto the PCI bus, for storage in the host computer's memory, or can be passed to other devices on the PCI bus such as disk controllers or S/VGA adapters.

Display

Image display is not a function of the PIXCI® SV4 imaging board, however it is discussed here since many applications require image display. The PIXCI® SV4 imaging board relies on an S/VGA board for image display. Depending on the VGA adapter, 24 bit RGB color images, 16 bit S-Video color images, or 8 bit monochrome images may be displayed. The full, scaled, or cropped image may be placed anywhere on the VGA screen.

Color or monochrome image data can be passed across the PCI bus directly to the S/VGA for live video-in-a-window display, depending on the S/VGA adapter. With a fast processor, fast PCI bus, and fast S/VGA adapter, 30 frame per second color image data may be displayed. Some S/VGA adapters provide graphics overlay on the live video.

Features

- Composite/S-Video/NTSC/RS-170/CCIR/PAL to YCrCb (4:2:2).
- Square Pixel and CCIR601 Resolution for NTSC and PAL Formats.
- Chroma Comb Filtering.
- Programmable Horizontal Scaling and Vertical Scaling.
- Programmable Cropping.
- Vertical Scaling Line Store.
- Programmable Temporal Decimation for Reduced Frame Rate.
- Programmable Hue, Brightness, Saturation, and Contrast.
- Double Frequency Oversampling A-D Converters.
- Three Video Input Mux.
- NTSC/PAL Auto Detect.
- Automatic Gain Control.
- PCI Bus Master with Burst Mode for 132 MB/S Transfers.
- Packed or Planar Format.
- 512 Pixel Buffer to PCI Bus.
- Real-Time Digitization to PCI Bus and/or S/VGA Memory.
- Real-Time Digitization to Motherboard Memory.
- 4 Input Triggers and 4 Output Triggers.
- Plug 'N Play Operation.
- Extensive Image Processing Programmer's Libraries.

The PIXCI® SV4 imaging board, for the PCI bus, is designed to take advantage of the power of the host computer. Applications which were once restricted by limited memory or processing power can now be easily accomplished with the PIXCI® SV4 imaging board and a compatible PCI computer.