Processing real-time stereo video for an autonomous robot using disparity maps and sensor fusion

Donald Rosselot and Ernest L. Hall
Center for Robotics Research
Department of Mechanical, Industrial, and Nuclear Engineering
University of Cincinnati
Cincinnati, OH 45221-0072

ABSTRACT

The Bearcat "Cub" Robot is an interactive, intelligent, Autonomous Guided Vehicle (AGV) designed to serve in unstructured environments. Recent advances in computer stereo vision algorithms that produce quality disparity, and the availability of low cost high speed camera systems have simplified many of tasks associated with robot navigation and obstacle avoidance using stereo vision. Leveraging these benefits, this paper describes a novel method for autonomous navigation and obstacle avoidance currently being implemented on the UC Bearcat Robot. The core of this approach is the synthesis of multiple sources of real-time data including stereo image disparity maps, tilt sensor data, and LADAR data with standard contour, edge, color, and line detection methods to provide robust and intelligent obstacle avoidance. An algorithm is presented with Matlab code to process the disparity maps to rapidly produce obstacle size and location information in a simple format, and features cancellation of noise and correction for pitch and roll. The vision and control computers are clustered with the Parallel Virtual Machine (PVM) software. The significance of this work is in presenting the methods needed for real time navigation and obstacle avoidance for intelligent autonomous robots.

Keywords: Autonomous, robot, stereo vision, disparity map, sensor fusion, parallel computing

1. INTRODUCTION

The general problem of designing a machine for real time navigation and obstacle avoidance in an arbitrary environment is ongoing. This problem is often described as a problem in artificial intelligence, fuzzy logic, sensor fusion, intelligent control, ad infinitum. When dealing with numerous, noisy, conflicting, incomplete, and uncertain information from multiple streams, designing robotic computer systems and algorithms for autonomous navigation and obstacle avoidance is non-trivial. This paper discusses the Disparity Map, and their processing to extract information. The algorithm developed here can answer the simple but important questions such as "How large is the object in front of the AGV?", "How far in front of the AGV is the object?", "At what angle is the object, is it directly in front of the AGV or off to one side", "What height is the object?", "What is the slope of the terrain in front of the AGV?, and "Is there a hole in front of the AGV?". This algorithm cancels information from the floor, which is required to produce clear information about objects in the field of vision, and it also compensates for tilt (roll and pitch) which is required so that the floor is not mistaken for objects and objects can be clearly identified in a tilting situation. Matlab sample code is available at: http://www.ececs.uc.edu/~rosseldw/RobotStereoVision.pdf. Sample disparity maps and images are available at: http://www.ececs.uc.edu/~rosseldw/RobotStereoData.zip.

2. THE CUB SENSOR SYSTEM AND SENSOR FUSION

The sensors included on the Cub for obstacle avoidance are a stereo vision system by Point Grey Research, a SICK LADAR scanner, a tilt sensor, and 2 color video cameras. Refer to Figures 1 and 2.

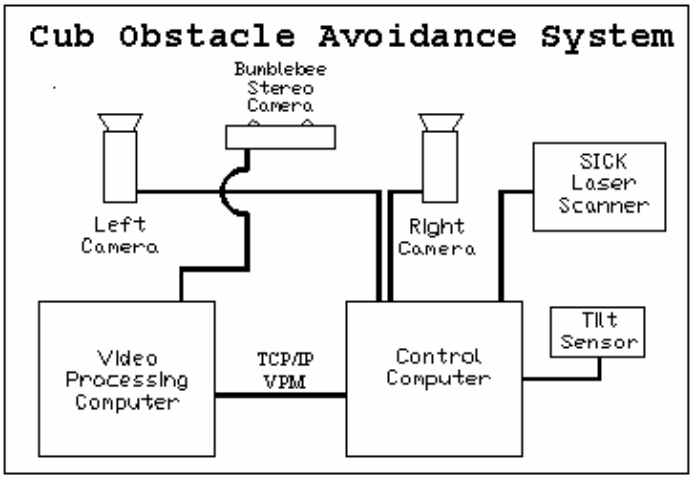


Figure 1. Obstacle Avoidance Hardware

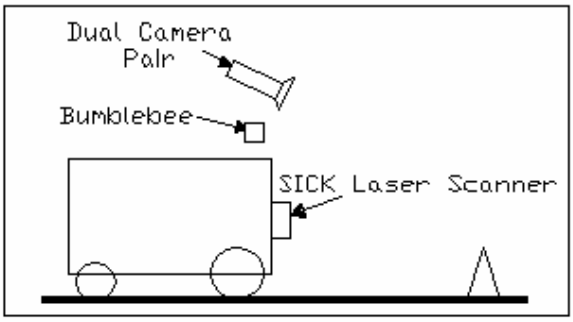


Figure 2. Sensor Arrangement

The stereo vision system adopted on the Cub is the Bumblebee by Point Grey Research. This is a packaged system that includes two pre-calibrated digital CCD cameras with a baseline of 12cm, and a C/C++ software API for sophisticated stereo vision processing. It will acquire and calculate disparity maps at 15 frames per second on a 3 GHz Intel Pentium for input into our obstacle avoidance algorithm. The Videre Design/SRI stereo vision system was also considered [1] for this application.

To put the current Cub AGV processing needs into perspective, a 640x480x8bit image acquired at 15 frames per second produces *4.6 Mbytes of data a second*. The Point Grey system uses the Single Instruction Multiple Data (SIMD) features of the Intel Pentium architecture to enable this large data throughput. As you may imagine, the vision processing software uses much of the CPU cycles available on the stereo vision computer. A TCP/IP connection and Parallel Virtual Machine (PVM) [2] software is used to create a two machine cluster to enable rapid communication, and processing of the multiple data pipelines. PVM is a message passing software system available freely at Oak Ridge National Laboratories that allows heterogeneous architectures to be pooled together into a virtual supercomputer. A great background text on implementing parallel architectures is Beowulf Cluster computing with Windows [3]. There is a four page introduction to PVM (http://www.csm.ornl.gov/pvm/intro.html) and the binaries for windows are available at(http://www.csm.ornl.gov/~sscott/PVM/Software/ParallelVirtualMachine3.4.3.zip).

The algorithm presented here to processes this and other information pipelines attempts to efficiently extract the relevant information from this data onslaught only what is required for robust and savvy obstacle detection. Although computers will continue to be developed that are faster and more sophisticated, algorithms will continue to be developed to increase the overall intelligence of AGV, and in turn the processing power required to deal with it. AGV's available today are multi-processing, with several processors on board. The Cub AGV for example, has a processor inside the SICK LADAR, the Galil motor controller, the cameras, the GPS system, and of course the computers. Enabling communication and parallel processing with PVM is just the next step in the evolution of existing communication protocols between processors such as *IEEE 1395* (Bumblebee Camera), *USB* (Camera Pair), *serial* (LADAR), and *TCP/IP* (motor controller).

The fusion of sensor data from these multiple data pipeline for obstacle avoidance and line following is designed in a simple but effective manner. The fusion between the tilt sensor and the stereo image data will be discussed in some detail below. Data from the camera pair is processed using the freely available open source OpenCV library [4], performing tasks such as edge detection, and line fitting with the Hough Transform. These methods are used to verify objects detected by the stereo vision system by fusing the information into a 3D world coordinate system. For example, if the stereo system detects an obstacle at a location, the camera pair should be detecting edges or contours in that space since the camera angle of the pair is known. The LADAR will also detect objects and is also a voting member. As this system is still under development, details are not solid on the voting scheme, and will certainly require tuning for various terrains.

3. STEREO VISION DISPARITY MAP PROCESSING

3. 1 The disparity (depth) map image

A disparity map or "depth map" image is an efficient method for storing the depth of each pixel in an image. Each pixel in the map corresponds to the same pixel in an image, but the grey level corresponds to the depth at that point rather than the gray-shade or color.

An experiment to demonstrate disparity would be to analyze two photos taken in a room a foot apart to create a crude stereo image pair. Print the photos on transparencies and lay them on top of each other and look for corresponding points (such as the corner of a garbage can). Objects nearer will have greater separation (this is the disparity), and objects very far away will line up very close (they will have less disparity). In fact, this is how stereo vision algorithms work. The computer attempts to match every pixel in an image with every pixel in the other using a correspondence algorithm. This will always be imperfect due to occlusions, poor texture (think of snow blindness), etc but works amazingly well quite often, as does the human visual system.

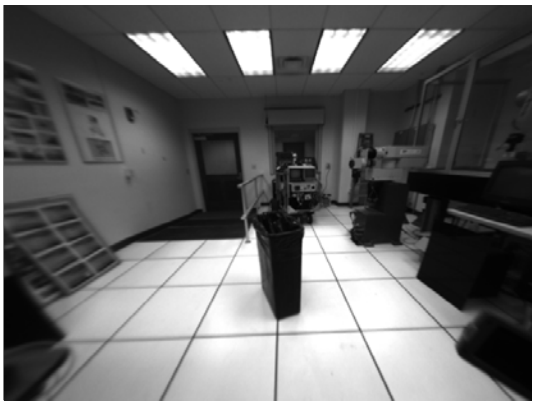


Figure 3. Indoor image of garbage can.

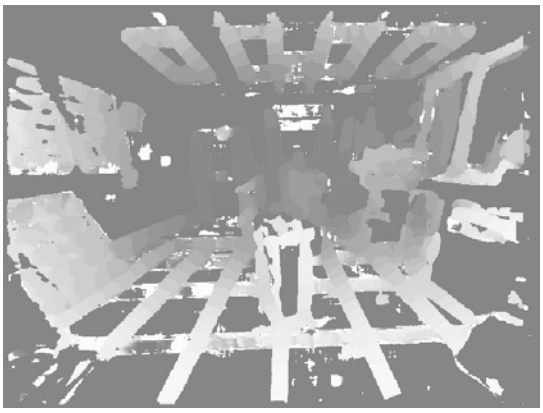


Figure 4. Histogram equalized disparity map**.**

Viewing a gray level histogram equalized disparity map image, objects that are lighter are close, and darker object are farther away. To determine the depth, two images are processed to produce a disparity map and since there are two images, the disparity value at each pixel is referenced to one image. Figure 3 is an image of a garbage can inside the robotics lab in the middle of the floor. Note in the disparity map in Figure 4, that the garbage can is readily visible in a light shade of gray. These images are directly from the point grey Bumblebee stereo camera system. The theory and applications of stereo processing is a huge area or research in and of itself, but the interested reader is referred to [5] and [6] for an excellent synopsis and overview of the state-of-the-art. Creating quality depth maps at 15 fps is not trivial, and the Bumblebee enables research focus on other tasks. In Figure 4, note the lines on the floor going from the foreground to the background, and how they darken as their depth increases. Data from the floor can be problematic when processing the maps and will be discussed below.

2.2 Point Grey disparity maps

Close reading of the Point Grey (PG) Stereo Vision Manual [7] included with the Bumblebee Stereo Camera system will reveal the finer points of their disparity maps. The PG system will produce disparity maps in either an 8 bit or 16 bit format in the portable grey map (.pgm) file format. The 8 bit image is an unsigned integer 8 bit format with 256 levels grayscale and can be viewed with MatLab or the freely available IrfanView viewer program (www.irfanview.com). Matlab 6.5 (13) or above is required to read .pgm files.

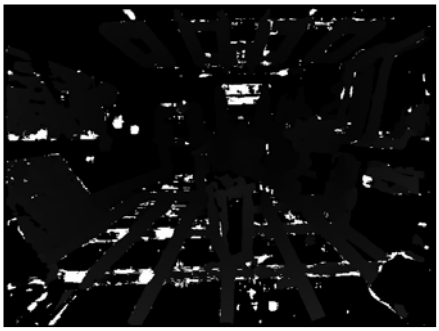


Figure 5. Raw 8 bit disparity map.

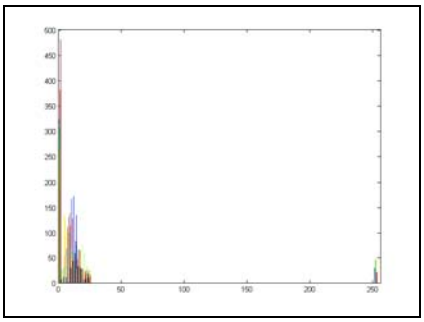


Figure 6. Histogram of raw 8 bit map.

The 16 bit depth images are in fixed point format, the high byte being the integer portion, and the low byte being the decimal portion. In reference to their 8 bit format, Figure 4 was histogram equalized using the PG function triclopsSetDisparityMapping(). This function creates a disparity map that is "balanced" visually for viewing. The actual image produced for extracting depth is quite different and shown in Figure 5, and it's histogram is show in Figure 6. Note that all of the information is stored in the first 26 levels as may be inferred from the histogram in Figure 6. Because virtually all of the pixels values occur from 0 to 26, it is very dark. In general, there may be more pixel values (gray levels) under different circumstances such as outdoor scenes, or changes in camera focal length. Level 240 to 255 is reserved for invalid bits, and 0 is used for points at infinity. The white that you see in Figure 5 is mapped to invalid pixels, pixels that could not be mapped for various reasons. These values are ignored in our analysis. Zero values also need not be processed.

When producing a disparity map in 16 bit format, an attempt at direct display creates the archetypical and bizarre results as in Figure 8. Figure 8 was produced from the image in Figure 7. These 16 bit maps can be manipulated with Matlab code to obtain the histogram equalized disparity map in Figure 9 below. Note that the garbage can is clearly visible in light gray, and the box, which is in the background and is a darker shade of gray.

```
A = imread ( 'disparity16.pgm');
data = bitand(A,255) ;                  % Strip off Fractional portion
A = uint8(data);                        % Get back to 8 bit format
figure, imshow(A)                       % Show the image before histogram
J = histeq(A);                          % Perform histogram equalization
figure, imshow(J)                       % Show the image after histogram equalization
```

Figure 7. Image of garbage can and box

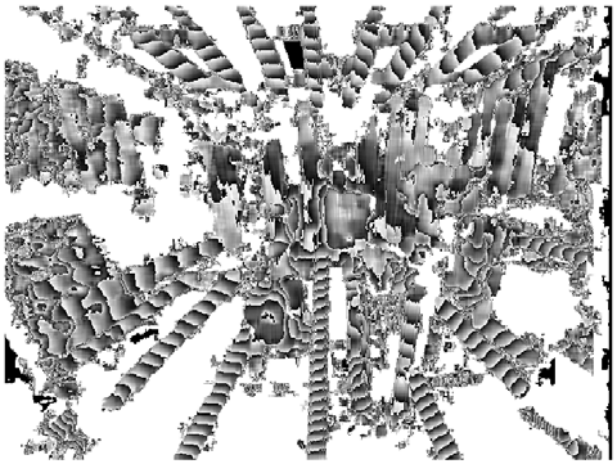


Figure 8. Raw 16 bit map image

Figure 9. 16 bit disparity Map

3. EXTRACTING OBJECT POSITION AND DEPTH INFORMATION FROM DISPARITY MAPS

3.1 Preliminaries: Disparity processing algorithm

The information contained in the disparity maps may be interesting to look at, but it is useless for AGV obstacle avoidance until it is processed to obtain object position and depth information. Without loss of generality, consider a disparity map is 480 pixels high, 640 pixels wide and 256 (0-255) levels in gray-level depth. Referring to Figure 10 below, visualize a depth map as 256 separate images, each depth image will have information about the image at that depth. For example, if the garbage can is *10 feet* from the camera, it might have several pixels at a disparity level 15. If it were at *14 feet* it might have several pixels at a disparity value of 11, (or found on plane 11). Recall that larger disparity values correspond to closer objects. Depending on depth resolution and object size and orientation, objects will generally appear at two or more image depth planes.

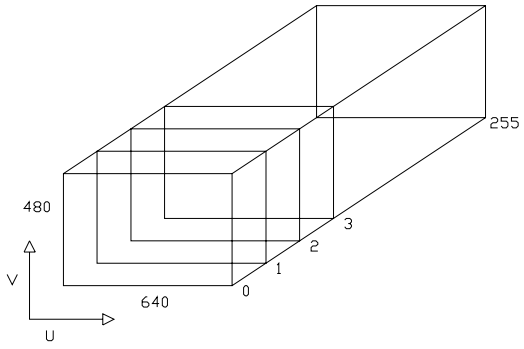


Figure 10. Image plane levels

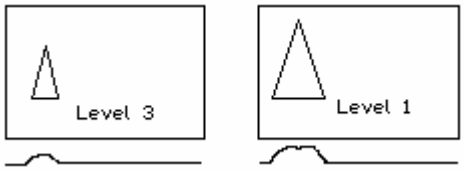
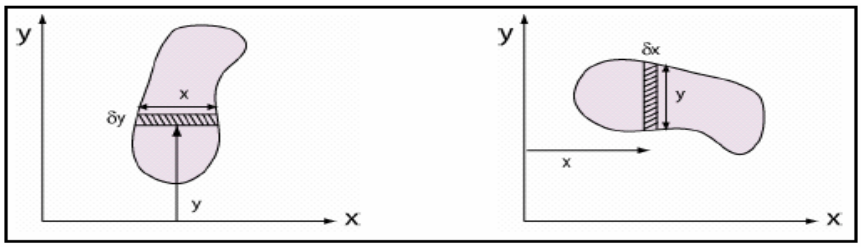


Figure 11 a. Level 3. Figure 11 b. Level 1.

Assume an image of a cone. Since level 3 (level, plane and disparity value are used interchangeably here) is nearer, the front surface of the cone may produce pixels as in Figure 11 a. Level 1 is farther away and may produce pixels similar to Figure 11 b. By summing and plotting the columns, curves are obtained similar to those shown below the figures. By finding the peak, the area, and the center of area under these curves, the size and position of the object can be determined with good precision. Finding the peak is trivial. Referencing Figure 10, the area and center of area are found as follows:



[8]

Figure 12. Calculating an area center.

First find the area: $A = \int y dx$

Second, find the first moment of the area:

$$M_{yy} = \int y_1 \, x \, dx$$

$$M_{xx} = \int x_1 \, y \, dy$$

Finally, the coordinates for the center of area are found using the equations:

$$\bar{x} = \frac{M_{yy}}{A} = \frac{1}{A}\int x \, y_1 \, dx$$

$$\bar{y} = \frac{M_{xx}}{A} = \frac{1}{A}\int y \, x_1 \, dy$$

Rather than processing disparity maps, a 3-D point cloud could be created from the pixels by using the formula Z=f*B/d, where Z is Depth, f is focal length, B is the baseline of the stereo camera, and d is the disparity value from the disparity map (depth map). However, the point cloud data would have to be processed, such as projecting the 3D data onto a plane resulting in several extra steps. To calculate actual X and Y position using the u and v coordinate from the disparity map, the formulas are X = uZ/f, and Y = vZ/f. The formula Z=f*B/d is used in this algorithm to determine the depth of each plane.

3.2 Specifics: Disparity processing algorithm

In practice, the process of breaking an image out into 255 image planes and summing by rows and columns would prove inefficient computationally. The task-at-hand is to take these concepts and produce an algorithm that is fast and efficient. Taking one column of data from any 480x640 image "A", will produce a 480x1 array. Performing a histogram count on this data in the full range of pixel values (0-255) results in a 256x1 array. Each of these arrays is put into a column to produce an array "X", 256x640.

```
%% ** Column analysis, No cancellation
A = imread ( 'depth.pgm');        % Load the image
[m,n] = size(A);                  % Get the size of the image
X=zeros(256,n);                   % Create an array to hold results
for i = 1:n                           % For every column.
       dataCol=A(:,i);               % Get the column of data:
       count=histc(dataCol, 0:255);  % get the hist. count of each number(0-255)
       X(:,i) =  count;               % and put it in a column of X

end                                  % X is the output, a 256x640 matrix
figure, plot(X(23,:))
```

The X array contains information about the number of occurrences of each disparity value across the X axis (the width of the image). Plotting each pixel across this axis, would produce 256 plots. A bit-shift of the data could be used to reduce the resolution down to decimate the calculation burden, but as it turns out, this is not necessary. Note that in Figure 6, the information is contained in only the first 26 disparity values which would require only 26 plots.

Direct analysis of these 26 data sets is great in theory, but in practice the noise produced by the data from the floor is so great, real objects cannot generally be distinguished. Refer to Figure 13 below. Note that the data in these figures were scaled for better viewing by multiplying it by 6, so that X and Y data could be included on the same plot. These plots were produced by processing the disparity map from Figure 5. Note that peaks along the Y axis is the height of objects found, with the Y axis direction reversed (Y axis positive is down). Therefore, the spike near the top is the signal from the floor, and the signal between the 250 and 300, is our object data. Looking along the X axis (X axis positive is right), it is nearly impossible to distinguish where the object might be. There are several peaks that correspond to the lines in the floor (Figure 13) that are giving extraneous peaks.

A strategy to remove the floor data is to use the information from the Y direction and determine the height of the objects producing the signals. By removing the signal from the floor, amazingly clear signals produced by the actual objects-of-interest can be seen. To cancel the floor data background noise at (for example) level 22:

```
% Background Cancellation at level 22
k1=23;  % add 1 for level 22. Matlab starts at 1, pix starts at 0!
for i = 420:450                      % Remove the floor data
    pixKill = find(A(i,:) == k1);    % Find background pixels in row to cancel
    A(i,pixKill) = 0;                % Cancel the background
end
```

Figure 14 is an example of this process. It can be seen that the floor data (yellow dotted line) has been drastically attenuated. The signal centered at 325 on the X axis (red solid line) that appears poorly attenuated is actually an initial signal from an object. At disparity value 19 (Figure 15), it is extremely clear that there is an object at this location. Note once again the attenuated signal is shown with dotted yellow lines.

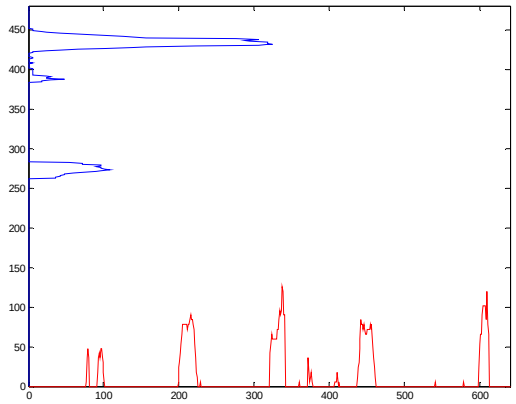


Figure 13. Floor signal

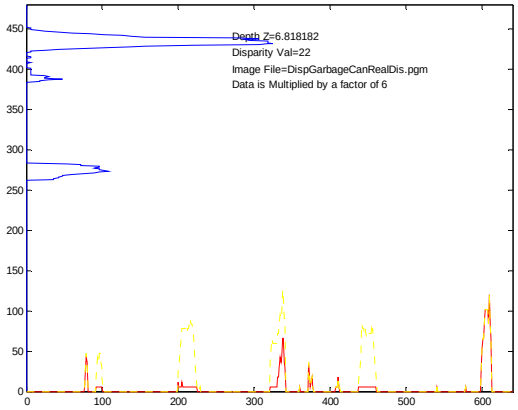


Figure 14. Floor Signal Cancellation

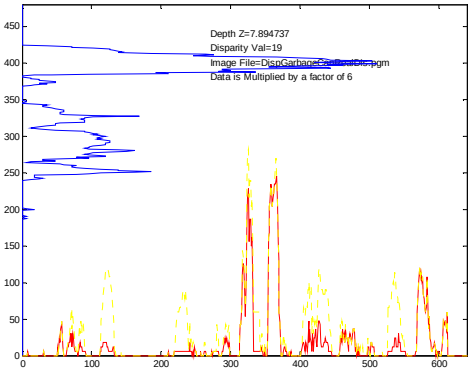


Figure 15. Obstacle Data

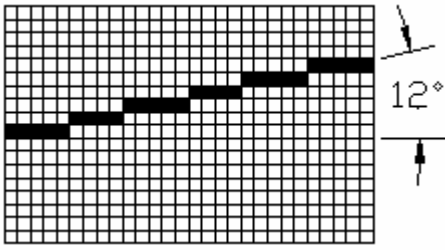


Figure 16. Tilt Compensation

A close look at the code snippet above demonstrating cancellation reveals the numbers 420:450. This is a method to cancel the data at this pixel value in those rows. This is a simplistic example for demonstrating the concept in a specific example, and this actually works well on a calibrated system on level ground, with a fixed camera focal length. But a more complicated algorithm must be used in practice to allow for roll, pitch, and changes in camera focal length. The cancellation must be done by image column in practice. This is computationally more expensive, but it is a price well worth paying. Imagine trying to walk with your inner ear unable to determine your head orientation. By using a tilt sensor and cancellation by columns, floor signal recognition and cancellation is possible. Referring to Figure 16, a 12º tilt would require a cancellation in the blacked out areas, which is done relatively efficiently while the columns are being processed. A Matlab code snippet is shown below that demonstrates this method with the features mentioned above.

```
%% ** Column analysis with cancellation
FloorIncrement=10;  % This may change with forward/back tilt (pitch)
Spread=30;          % This should remain constant, Spread=Floor noise width
TiltFactor=0;       % This will change with tilt, varies with roll, number of
                    % pix to move base up/down per 5 pix over, (5 may change)
X=zeros(256,n);         % Create the Matrix to hold data
for i = 1:n             % n is the width of the Pic
    dataCol=A(:,i);     % Get each column of data: note we are using data,
                        % m x n row/column matrix ie (640x480, 16 levels
    Base=220;           % Good for DispGarbageCanRealDis.pgm  This will change
                        % with forward/back tilt, and focal length, must be set
                        % inside this loop with real-time data
    % Base=250;           % Good for genWdis5-60depth.pgm
    for killVal = 2:25    % Done inside each Column to
                          % compensate tilt. Going across rows would be faster
        killZone = (dataCol(Base:Base+Spread))'; % Transpose to a row vector
        killPix = find(killZone == killVal);     % Find the value to cancel
        dataCol(Base + killPix)=0;            % Line up killZone to dataCol and
                                              % cancel floor pixels
        Base = Base + FloorIncrement;
        if(Base >= m - Spread)                    % Stay within bounds
            break
        end
        if((TiltFactor ~= 0) && (mod(i,5) == 0))  % We have moved 5 pixels,
                                                  % compensate base with tilt
                                                  % factor
            Base = Base + TiltFactor;             % Adjust Base for tilt
        end
    end
    count=histc(dataCol, 0:255);      % get the count of each number (0-255)
                                      % in each column of the Pic. note 0-255
                                      % occurs when code is changed
    X(:,i) =  count;                  % and put it in a row of X, each row
                                      % represents a depth, and the value across
                                      % the row are the number of
 end                                  % times a pixel was found at that depth.
```

4. CONCLUSIONS

The overall obstacle avoidance architecture for the University of Cincinnati AGV robot was discussed. A robust algorithm was presented for the processing of disparity maps in real time. The algorithm features roll and tilt compensation and cancellation for floor noise that is simple, fast, and effective.

REFERENCES

1, K. Konolige and D. Beymer, *Small Vision System User Manual*, 2003

2. The Oak Ridge National Lab, "*Parallel Virtual Machine Software*", Computer Science and Mathematics Division, http://www.csm.ornl.gov/pvm, 2004, Note: news group at: comp.parallel.pvm.

3. T. Sterling, *Beowulf Cluster Computing with Windows*, The MIT Press, Cambridge, 2001.

4. Intel Inc., "*The OpenCV Open Source Computer Vision Library*", http://www.intel.com/research/mrl/research/opencv

5. M.Z. Brown, D. Burschka, and G.D. Hager, "Advances in computational stereo", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **Vol. 25**, No. 8, August 2003

6. D. Scharstein, R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms", *International Journal of Computer Vision* , **Vol. 47(1/2/3)**, pp 7–42, 2002.

http://www.videredesign.com

7. Point Grey Research, "Triclops StereoVision System Manual Version 3.1" 2003
http://www.ptgrey.com

8. A.H. Techet, *"Hydrodynamics for Ocean Engineers"*, Class Notes,
http://web.mit.edu/13.012/www/handouts/Reading3.pdf