Predictive vision from stereo video: Robust object detection for autonomous navigation using the Unscented Kalman Filter on streaming stereo images

Donald Rosselot, Mark Aull and Ernest L. Hall
Center for Robotics Research, School of Dynamic Systems
University of Cincinnati, Cincinnati, OH 45221-0072

ABSTRACT

A predictive object detection algorithm was developed to investigate the practicality of using advanced filtering on stereo vision object detection algorithms such as the X-H Map. Obstacle detection with stereo vision is inherently noisy and non linear. This paper describes the X-H Map algorithm and details a method of improving the accuracy with the Unscented Kalman Filter (UKF). The significance of this work is that it details a method of stereo vision object detection and concludes that the UKF is a relevant method of filtering that improves the robustness of obstacle detection given noisy inputs. This method of integrating the UKF for use in stereo vision is suitable for any standard stereo vision algorithm that is based on pixel matching (stereo correspondence) from disparity maps.

Keywords: Autonomous navigation, Unscented Kalman Filter, stereo vision, XH-map algorithm, Disparity Map

1. INTRODUCTION

An overview of the UKF will be given, specific implementation details with the nonlinear equations of stereo vision and the results of using the UKF with stereo vision. Finally a review will be made of the XH-map algorithm which addresses the problem of the robust and rapid detection of obstacles in 3-space and accurate mapping to a 2D map using live stereo vision video.

2.0 THE KALMAN FILTERING AND THE UKF

2.1 Stereo object detection noise

Obstacle detection from streaming stereo images is noisy. Stereo image pairs arrive at 15 frames per second or faster and each pair must be processed to create a disparity map using a correspondence algorithm. This part of the process is done quite effectively by the Bumblebee camera software, but with a still camera and even light there is significant difference between the disparity maps of successive frames. Any algorithm that uses streaming disparity maps with today's technology will be sourced from a noisy signal.

Once an objected is detected with the XH-map algorithm (or any standard algorithm that is computed from disparity images) the well known and non-linear Thales formulas can be used to find an object in Cartesian co-ordinates relative to the camera.

$$Z = f * \frac{B}{d} \tag{2.1}$$

$$X = u * \frac{Z}{f} \tag{2.2}$$

$$Y = v * \frac{Z}{f} \tag{2.3}$$

Where

f	is a constant and is the effective focal length in pixels
B	is a constant and is the baseline distance in meters of the camera (The distance between lens centers)
d	is the disparity in pixels matched in the disparity map
(u, v)	is the pixel location in the 2D image
(X,Y,Z)	is the real world 3D position

Since the obstacle position X, Y, and Z is sourced from d, u, and v via disparity maps at several frames per second, the detected obstacle position will be noisy. Kalman filters of many types exist and are widely used to reduce noise in a variety of systems. The first problem in addressing the noise in any situation is finding which Kalman filter is a good and relevant solution to the problem-at-hand.

2.2 Overview of Kalman filtering noise reduction

A good introduction of the Kalman filter can be found in Welch & Bishop [1] and their website [2] has a set of examples of learning tools with Matlab code.

The original Kalman filter [3] required a linear state and observation model and was soon extended to the Extended Kalman Filter (EKF) to address state and/or measurement models that were non-linear. Partial derivatives of the state and/or measurement models are used to estimate the current mean and covariance. The EKF suffers from the problems that the random variables of mean and covariance lose their normal properties, and getting an approximation of the models with partial derivatives of the models is often difficult, requires a lot of computation time, or is impossible. To address the shortcomings of the EKF, Julier et al. [4] created the UKF, a simpler, faster, and more accurate filter which makes use of the unscented transformation for the mean and covariance. Several other variations of the Kalman filter exist, but the UKF was chosen as a good choice for this implementation for its relative simplicity, accuracy and computational speed for real-time object detection from streaming stereo video.

2.3 General equations of the UKF

The section follows very closely with Simon's chapter on the UKF [5].

With an n-state discreet-time nonlinear system of

$x_{k+1} = f(x_k, u_k, t_k) + w_k$ The general state model

(2.4)

x_{k+1} is a function of the current state x_k, control input u_k, time step t_k, and process noise w_k

$y_k = h(x_k, t_k) + v_k$ The observation model (or measurement equation) with observation noise v_k

(2.5)

The time update equations

First create sigma points $\hat{\chi}_{k-1}^{(i)}$ around x_k

$$\hat{\chi}_{k-1}^{(i)} = x_k + \hat{x}_{k-1}^{+} + \tilde{\chi}^i \quad i = 1,...2n \tag{2.6}$$

where

x_k The current state

$\hat{x}_{k-1}^{+} = 0$ Zero mean for the sigma distribution in our system

$\tilde{\chi}^i = (\sqrt{nP_{k-1}^{+}})_i^T \quad i = 1,...n$ where $(\sqrt{nP_{k-1}^{+}})_i$ is the $\underline{i}$ row of the matrix square root $\sqrt{nP_{k-1}^{+}}$

$$\tilde{\chi}^{n+i} = -(\sqrt{nP_{k-1}^{+}})_i^T \; i = 1,...n$$

Use the state equation to transform the sigma points into $\hat{\chi}_{k-1}^{(i)}$ vectors

$$\hat{\chi}_k^{(i)} = f(\hat{\chi}_{k-1}^{(i)}, u_k, t_k) \tag{2.7}$$

Combine the $\hat{\chi}_k^{(i)}$ vectors to obtain the *a priori* state estimate at time k.

$$\hat{\chi}_{\bar{k}} = \frac{1}{2n}\sum_{i=1}^{2n} \hat{\chi}_{\bar{k}}^{(i)} \tag{2.8}$$

Estimate the a priori error covariance with the process noise

$$P_{\bar{k}} = \frac{1}{2n}\sum_{i=1}^{2n} (\hat{\chi}_{\bar{k}}^{(i)} - \bar{\chi}_{\bar{k}}^{(i)})(\hat{\chi}_{\bar{k}}^{(i)} - \bar{\chi}_{\bar{k}}^{(i)})^T + Q_{k-1} \tag{2.9}$$

The Measurement update equations

Use the observation model to transform the sigma points into $\hat{y}_k^{(i)}$ vectors (predicted measurements)

$$\hat{y}_k^{(i)} = h(\hat{x}_k^{(i)}, t_k) \tag{2.10}$$

Combine the $\hat{y}_k^{(i)}$ vectors to obtain the predicted measurement at time k

$$\hat{y}_{\bar{k}} = \frac{1}{2n}\sum_{i=1}^{2n} \hat{y}_{\bar{k}}^{(i)} \tag{2.11}$$

Estimate the measurement covariance with measurement noise

$$P_{yy}^{\bar{}} = \frac{1}{2n}\sum_{i=1}^{2n} (\hat{y}_{\bar{k}}^{(i)} - \bar{y}_{\bar{k}}^{(i)})(\hat{y}_{\bar{k}}^{(i)} - \bar{y}_{\bar{k}}^{(i)})^T + R_{k-1} \tag{2.12}$$

Estimate the cross covariance between $\hat{\chi}_{\bar{k}}$ and $\hat{y}_k$

$$P_{xy}^{\bar{}} = \frac{1}{2n}\sum_{i=1}^{2n} (\hat{\chi}_{\bar{k}}^{(i)} - \bar{\chi}_{\bar{k}}^{(i)})(\hat{y}_{\bar{k}}^{(i)} - \bar{y}_{\bar{k}}^{(i)})^T \tag{2.13}$$

2.4 An implementation of the UKF for stereo vision

As the simplest case to investigate the use of the UKF for stereo vision, the state and observation models are constant. The state model can be kept constant and linear by assuming that the object is not moving. The observation model can be kept constant by assuming that the robot is not moving, but it is nonlinear per Equations 2.1 and 2.2. A moving object would require modification of the state model. For an example of a Newtonian system with position, velocity and acceleration see Simon[6]. A moving robot (camera) would require modification of the observation model with the addition of the frame-of-reference rotation and translation equations and can be found in Vince [7] and others.

The signals are simulated with arbitrary values of 147.7 for disparity and 17.5 for the u value (x pixel location) which correlates to an approximate Z and X position of 1.5 and 1 respectively. Both are corrupted by adding10% Gaussian noise.

The 2-state discreet-time system is

$$Z_{k+1} = Z_k + w_k \quad \textbf{The linear state model } Z \text{ position}$$
$$X_{k+1} = X_k + w_k \quad X \text{ position} \tag{2.15}$$

Note that the state remains constant between steps. Z and X will vary due to the noisy observation data.

$$d_{k+1} = \frac{f * B}{Z_k} \quad \textbf{The nonlinear observation model}. \text{ Disparity}$$
$$u_{k+1} = \frac{X_k * f}{Z_k} \quad u \text{ pixel position} \tag{2.16}$$

2.5 UKF Filtering of stereo data: Results and Discussion

In Figures 2.1 and 2.2 below the larger signal is the position before filtering, and the smaller signal is after filtering. The ground truth is the horizontal line at 1.4418. These figures demonstrate the effect of tuning the process noise R. In figure 2.1 R is set to .01 and in 2.2 R is set to 0.1. As discussed in Welch et al. [1], when the filter measurement variance is increased, it is "slower" to believe the measurements. After some tuning, R= 0.1 was found to be a good tradeoff between accuracy and response in this application. The Z axis standard deviations are 0.040 and 0.0105 before and after filtering respectively. The X standard deviations are nearly the same. The unfiltered and filtered data were both plotted as lines in Figures 2.1 and 2.2 to demonstrate visually the response of the filter. In Figures 2.3 and 2.4 the unfiltered data was plotted as points and the filtered data was plotted as a line to convey visually the signal detected in the noise. Based on these results and the efficiency of this implementation of the UKF algorithm (45 lines of Matlab code in the main loop), it is concluded that this is an effective method of increasing the robustness of obstacle detection. The Matlab code for this investigation and that was used to create the figures in this section can be found at the UC robotics website[8].

Figure 2.1 with R (process noise) set to 0.01. Figure 2.2 R is set to 0.1. 100 time steps

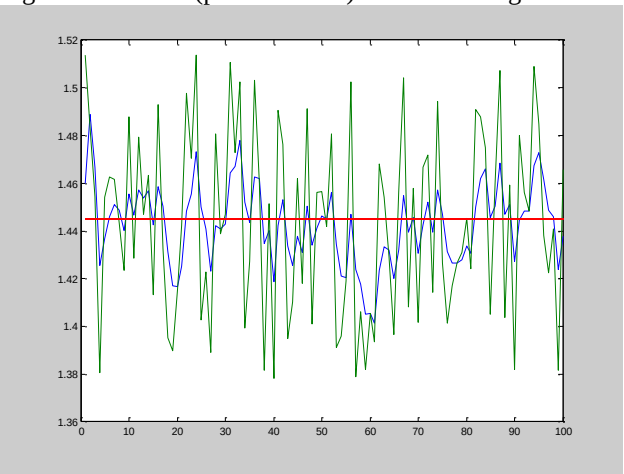

Figure 2.1

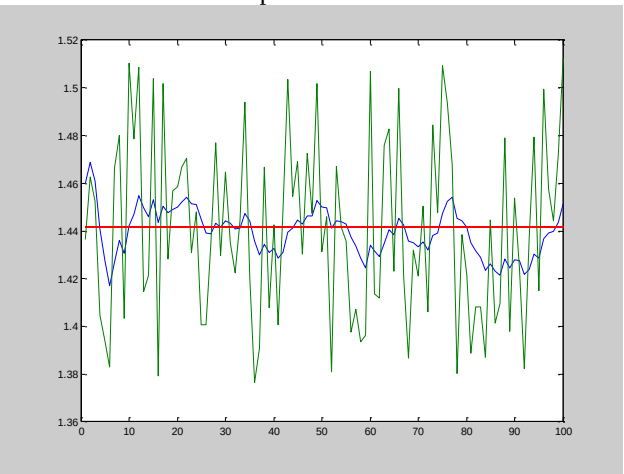

Figure 2.2

Figure 2.2 and 2.3 below are the Z and X position respectively. Dots are unfiltered, signal line is filtered, and the horizontal line is ground truth. 500 time steps, R=0.1

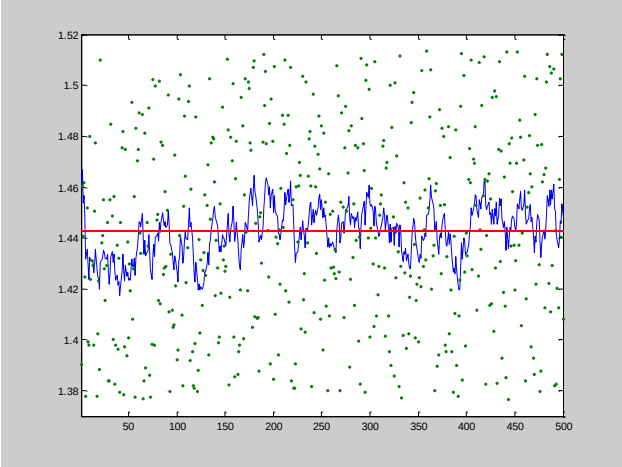

Figure 2.3

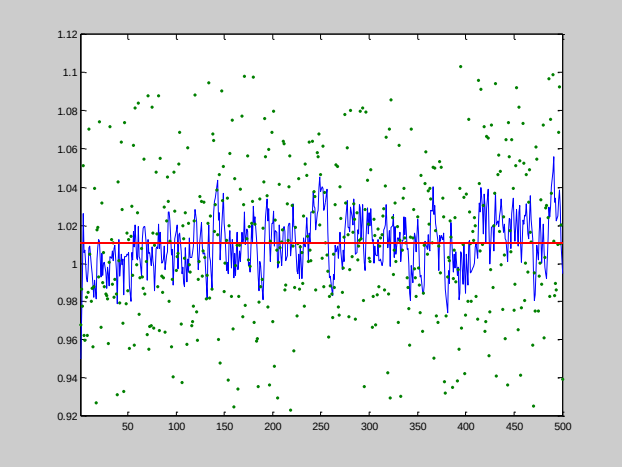

Figure 2.4

3. Video acquisition and XH-Map processing overview

3.1 Stereo video image frame processing overview

The stereo video acquisition system used is the Bumblebee stereo vision system by Point Grey Research (PGR)[9]. The loop function sequence from the PGR Software Development Kit (SDK)[10] to communicate with the Bumblebee and to continuously create disparity maps for this obstacle avoidance algorithm is the first five functions in the loop and is later discussed as GrabDisparityImage() and be summarized by:

Begin Loop
  digiclopsGrabImage(digiclops);
    Grab the image from the camera and store all information in the digiclops context
  digiclopsExtractTriclopsInput(digiclops, STEREO_IMAGE, &inputData);
    Extracts the data from the context and puts it into the inputData data structure.

triclopsPreprocess(triclops, &inputData);

Unpacks and rectifies the image and may do smoothing, and edge detection if requested.

triclopsStereo(triclops);

This function performs stereo processing on the image to produce the disparity map.

triclopsGetImage(triclops, TriImg_DISPARITY, TriCam_REFERENCE, &depthImg);

This function retrieves the disparity map image from the triclops context and places it into depthImg.

"Process obstacle avoidance algorithm using disparity map(depthImg)"

This is the XH-map Algorithm and is expanded and discussed in detail below.

End Loop

3.2 XH-Map processing overview

The core of the XH-map algorithm uses a disparity map from a stereo image pair and:

- Removes floor-ground background and other unwanted "noise".
- Separates the image into depth planes.
- Aggregates column information into smaller usable sets and sum those sets by depth to produce obstacle information at each depth.
- Uses that information to produce a "live" occupancy grid of objects from streaming video

The algorithm creates a calibrated obstacle map in real-time using a commercially available stereo camera system. Detecting obstacles using stereo vision in an application such as navigation for an Autonomous Ground Vehicle (AGV or robot) requires considerable processing efficiencies to handle live video at real-time rates. Although commercial camera systems can deliver "pretty good" disparity maps at fifteen frames per second or better, processing and performing useful algorithms such as obstacle mapping while maintaining high information velocities with a standard Pentium 4 computer requires resourceful algorithms and careful programming techniques.

This algorithm will 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?", and "At what angle is the object, is it directly in front of the AGV or off to one side". The XH-map algorithm creates a "floor plan" of obstacles. Height information of objects is purposefully lost in the 3D to 2D mapping.

4. XH-map ALGORITHM: EXTRACTING OBJECT POSITION FROM DISPARITY MAPS

4.1 XH-map algorithm overview

Figure 4.1 is one image from a stereo pair taken inside the UC robotics lab with a garbage can in the center of the floor. Figure 4.2 is a disparity map from this stereo pair that has been grey scale balanced (Histogram equalized) for better viewing. Consider a disparity map image 480 pixels high, 640 pixels wide and 256 (0-255) levels in gray-level depth. Visualize this map as 256 separate images, one for each gray level or depth plane. The depth planes can be thought of a slices in depth of the three dimensional world. Each slice will have information about the world at that depth. Figure 4.3 is an example of a depth plane taken from the disparity map of figure 4.2. Two depth planes or "world slices" were chosen at the depth of the garbage can and the pixels can be clearly seen. Depending on depth resolution and object size and orientation, objects will generally appear at two or more image depth planes. By summing each column or row at each depth plane, a profile function at that depth can be obtained as sketched in Figure 4.2. A large peak in the vertical profile usually is the "signal" from the floor, and peaks in the horizontal profile are often objects at that depth.

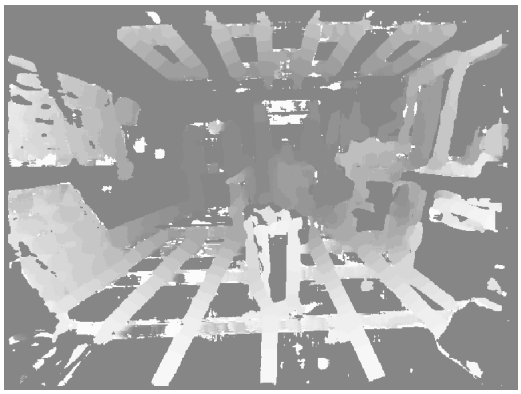

Figure 4.1 Indoor image of garbage can.

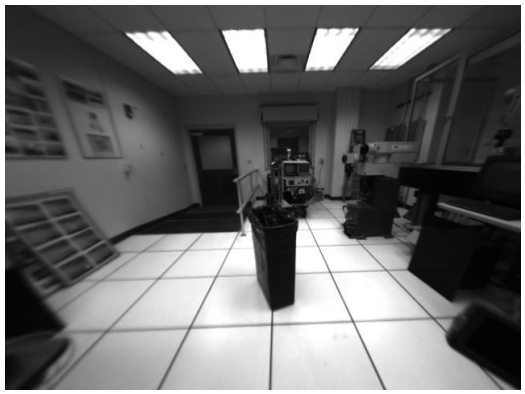

Figure 4.2 Histogram equalized disparity map.

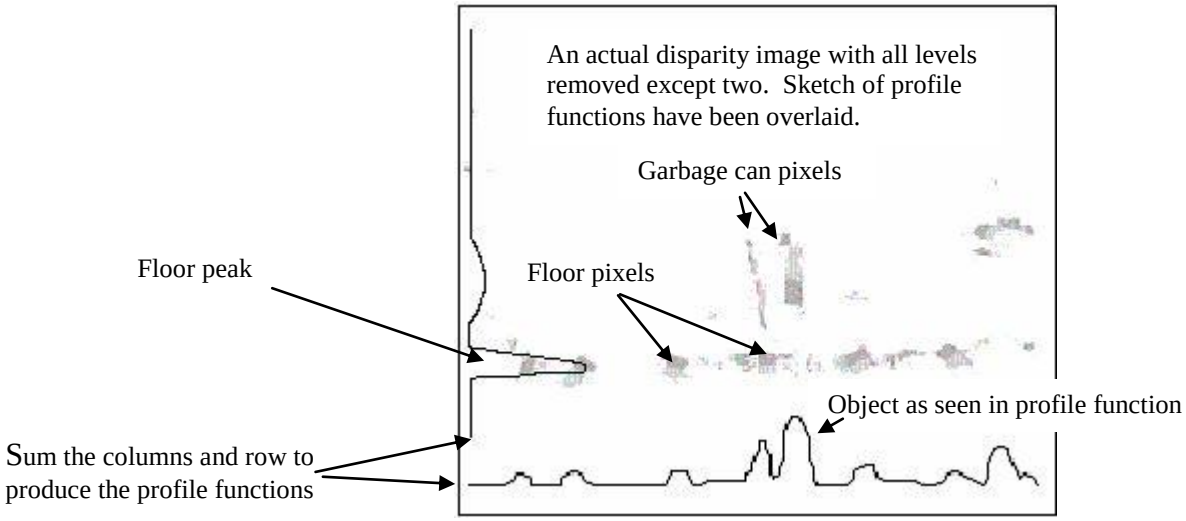

Figure 4.3 Actual depth plane from a disparity map

4.1 Floor Noise cancellation

Using the Matlab[11] Image processing library, Figures 4.4 through 4.6 below were created for the purpose of illustrating the concepts of Floor cancellation. By summing the rows (the Y axis) and the columns (the X axis) of a single depth plane (disparity value 22) Figure 4.4 was created from the (original) disparity image of Figure 4.2. It is difficult to determine where an object is by looking at the X axis. It appears that there may be 5 or more objects based on the signal peaks. These peaks correspond to the thin black lines on the floor as seen in Figure 4.1. Note that the Y axis direction is reversed (Y axis positive is down) as is common in image processing. The spike near the top along the Y axis is the signal from the floor. The strategy here is to remove the floor data by using information from the Y direction. During the calibration stage, the height of the floor is determined at each depth to enable removing that signal during runtime. By removing the signal at the floor, amazingly clear signals produced by the actual objects-of-interest can be seen.

Figure 4.5 (at disparity value 22) and Figure 4.6 (disparity value 19) are examples of this process. Note that larger disparity values correspond to closer distance. Figure 4.5 is at a level near the front surface of the garbage can and Figure 4.6 is in the plane of the garbage can. It can be seen that the floor data (yellow dotted line) has been drastically attenuated in Figures 4.5 and 4.6. The signal centered at 325 on the X axis (red solid line) that appears poorly attenuated is actually an initial signal from the nearest edge of the garbage can. At disparity value 19 (Figure 4.6), it is extremely clear that there is an object at this location. The signal before attenuation is shown with dotted yellow lines and is overlaid by the red attenuated signal.

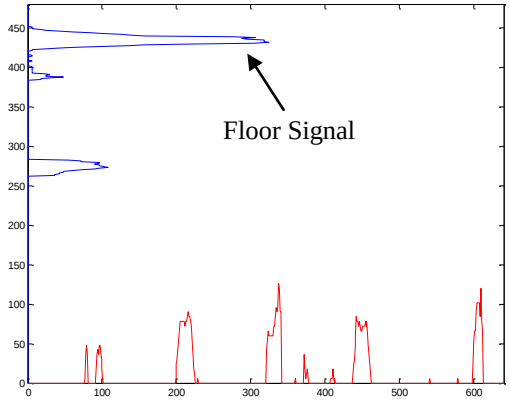

Figure 4.4 Floor signal

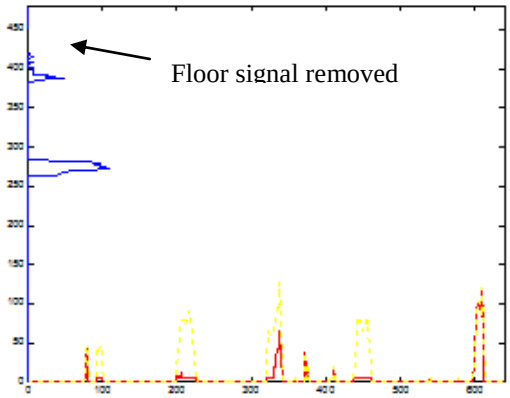

Figure 4.5 Floor Signal Cancellation

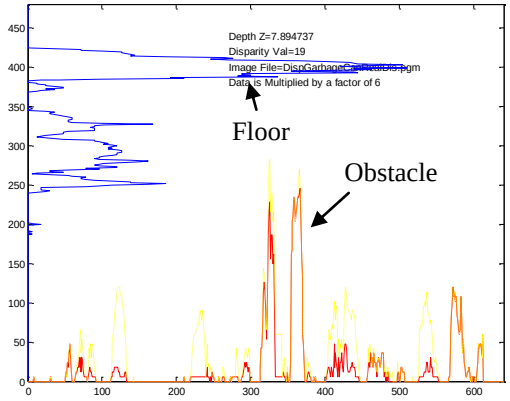

Figure 4.6 Profile function of obstacles and floor

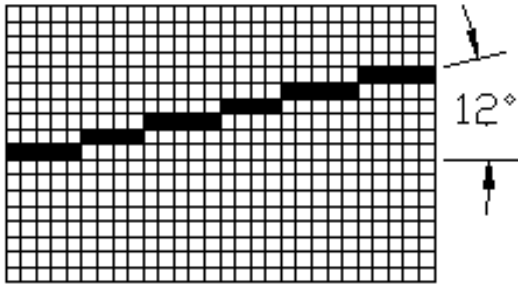

Figure 4.7 Tilt Compensation

4.2 Implementation of the XH-map algorithm

In practice, to find objects at each depth by summing rows and columns by the process of dividing an image into 255 image planes would prove inefficient computationally. The task-at-hand is to take these concepts and produce an algorithm that is fast and efficient. The algorithm was written in C++ and uses the Intel IPP[12] library and the Intel OpenCV[13] library.

The main loop is:

```
Begin Loop
  GrabDisparityImage (imgDispMapSrc)
  ippiRotateCenter_8u_C1R(imgDispMapSrc , imgDispMapRot, rotAngle, …)
  RemoveFloorBkGnd(imgDispMapRot, imgDispMap, FloorIncrement, Spread, Base)
  RemoveTopThird(imgDispMap)
  CreateAggregatedX-HistoMap(imgDispMap, XhMap)
  CreateOccupancyMap(triclops, XhMap)
End Loop
```

Roll compensation input from sensor

Pitch compensation input from sensor

This loop continually grabs stereo image pairs from the bumblebee video stream and produces a disparity image, processes those images and produces a "live" calibrated occupancy map in real time. Objects that are moving can be seen to move across the map, and as the robot moves forward, objects flow from top to bottom of map as you would expect. Each of these steps are now discussed in more depth.

1. GrabDisparityImage(imgDispMapSrc) Grabs a stereo pair from the video stream and outputs a disparity image imgDispMapSrc(). This function was discussed in detail above.
2. ippiRotateCenter_8u_C1R(imgDispMapSrc , imgDispMapRot, rotAngle, …) This function performs roll compensation based on input from the tilt sensor. It simply rotates the image if required to offset any rotation about the roll axis of the robot. Roll compensation is required to keep the floor-ground information from the cameras in the same relative position as determined at calibration. For example assume that the floor information at that depth was level at calibration, but now has been rotated 12 degrees as sketched in Figure 3.7 above. By detecting the rotation with a tilt sensor and counter rotating the disparity image 12 degrees the floor data will once again be at level and can be removed. Referring to Figure 3.6 below, roll is defined as α, pitch (forward/back tilt) is defined as γ. Although the process of removing floor background is covered in detail just below, the pitch compensation input is also data received from the tilt sensor and deserves mention here. A rotation about γ (pitch) will cause the image to translate in the Y dimension (appear to move up and down). Hence the position of the floor will translate in Y and must be accounted for. The variable "base" is the position in the image where floor cancellation begins and must be calibrated to offset pitch using information from the tilt sensor.
3. RemoveFloorBkGnd(imgDispMapSrc, imgDispMap, FloorIncrement, Spread, Base) Removes the floor information from the disparity map. Removing the floor data simply requires replacing the appropriate numbers in the appropriate rows with zeros to cancel their contribution when the numbers are binned by the histogram operation. This function was made extremely efficient by the Intel IPP function ippiLUT_8u_C1R(…), which will replace a single (or set) of values in an image in one function call. For floor cancellation, this function merely iterates through the 20 or so relevant depths and replaces the floor data with zeros so that the floor adds nothing to the column sum. Compare this 20 iterations with a direct search through a 40x640 section of image (they typical size need for floor cancellation) times 20 would require 512,000 iterations. The floor *Spread* typically encompasses forty adjacent pixel rows on each depth plane and moves up by *FloorIncrement* amount, typically nine rows of pixels with each iteration in increasing depth in the disparity map. These numbers are set during the calibration phase (described below). The numbers forty and nine are dependent on the properties of the camera such as focal length and of image resolution, both which never change in our application. *FloorIncrement* is also the amount the floor peak advances in units of pixels per increase in depth during the calibration stage. *Base* is the position (in units of pixels) in the disparity map to begin canceling the floor data. It is easily identifiable as the first peak in the histogram set of histograms created while iterating through the depth planes during the calibration phase. If the AGV pitches forward or aft, the tilt sensor information is sent to *Base* to maintain the relative disparity image position determined at calibration. The argument imgDispMapSrc is the input image, and imgDispMap is the output image.
4. RemoveTopThird(imgDispMap) Removes the top third of the image information by replacing the data with zeros. This function is made extremely efficient by the IPP function ippiSet_8u_C1R(…) which will set a region of an image to a specified value in one operation. Removing the top third can make the algorithm more robust in cluttered indoor environments when objects above a certain height can be neglected, such as on the current application of an AGV robot.
5. CreateAggregatedX-HistoMap(imgDispMap, XhMap) uses the IPP function `ippiHistogramEven_8u_C1R(…)` to sum columns at each depth in one operation. This is the heart of the algorithm and is extremely efficient, but not very intuitive and requires a bit of explanation. First and foremost is the speed requirement issue. For the current occupancy grid of 20x20, this function merely calls `ippiHistogramEven_8u_C1R` twenty times to aggregate 640 columns down to 20 (using a Region of Interest) and sum all twenty depths to produce the matrix required for the occupancy grid. This function contains only nine lines of code, five of which are definitions, and loops through 20 iterations. Compare this for example, with looping through all 256 depths and 640 columns or 256x640 =163,840 iterations. Since

this loop needs to run 12 times a second, it can ill afford to be inefficient. The input to this function is the disparity map image, and the output is an XH-map matrix as can be seen in Figure 4.11below. Consider a 640x480 image divided into 20 vertical columns by aggregating the information from sets of 32 columns (640/32=20) to produce something as represented by the topmost graphic in Figure 4.11 below.

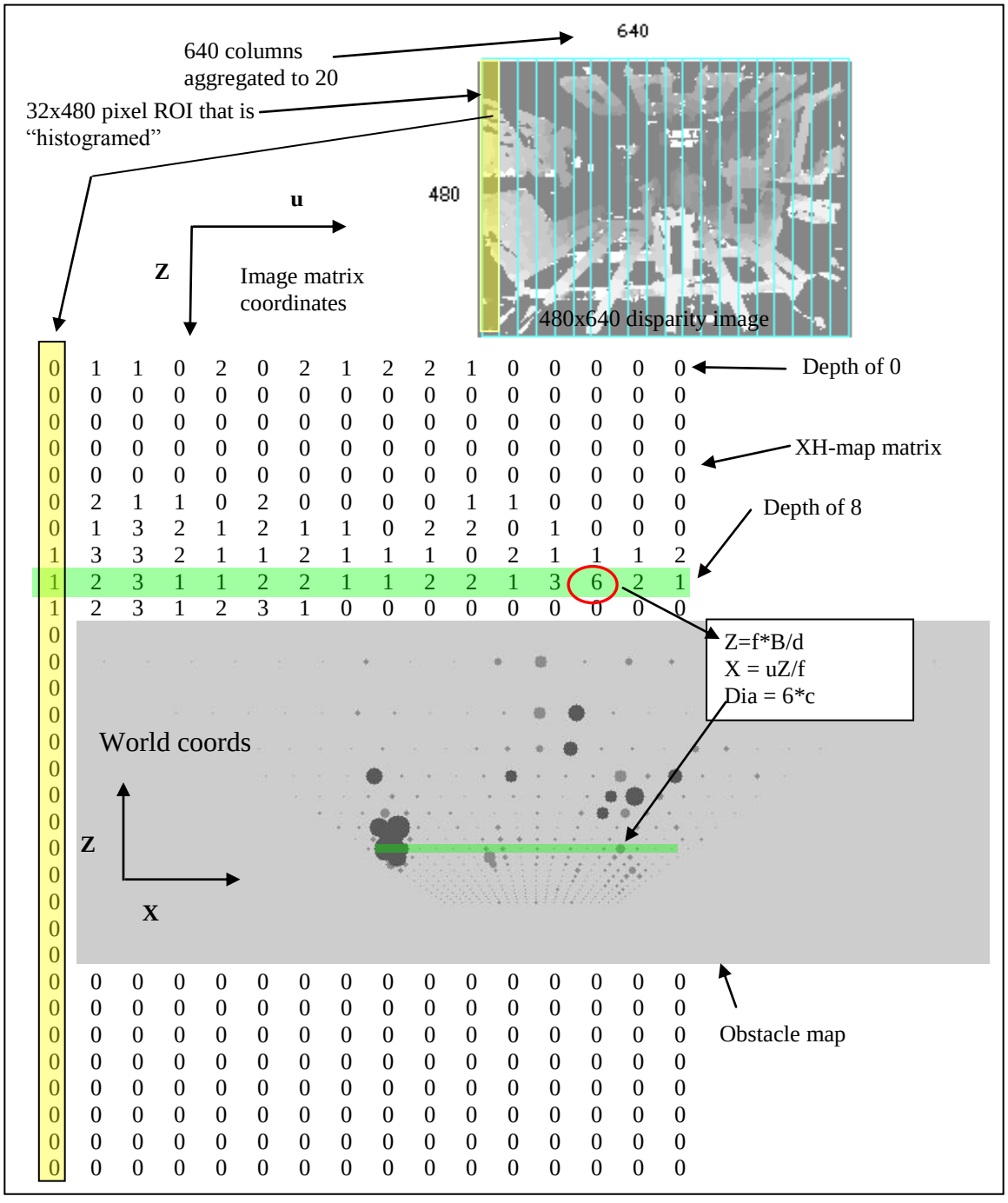

Figure 4.11 Creating the X-H Map

This 20x480 section of image contains values in the range of 0-255, each value representing a depth. Since we are only interested in values 4 thru 24 in our application (which scales to depths of about 1 meter to 5.3

meters), we will perform a histogram on this 32x480 section in the reduced range of 0-32 (to encompass 4-24) instead of the full range of 0-255. The histogram just bins the numbers, giving sums for the number of pixels that occur at that depth. If the histogram from each column is put into a column of an XH-map a matrix of numbers is obtained. Each column of the original disparity image map image is now transformed into an XH-map matrix of 20 by 32 numbers. Figure 3.11 contains only 16 columns rather than 20 merely so the graphic fits on one page. The X-Z coordinates of a number (for example, number 6 at depth 8 as circled in Figure 3.11) can be calculated by the equations Z=(f*B)/d, and X = (u*Z)/f and plotted as in the grey obstacle map in the Figure. The diameter of each obstacle is calculated by D=q*c, where q is the value at that point in the XH-map and c is a constant to give a good appearance to the obstacle map. Since q (6 in our example) is the number of pixels found at that depth and in that 32x480 area (Region of Interest or ROI) in the disparity map, it is a pretty good indication of the area or size of the obstacle. Since height information is collapsed in this process, a high and low obstacle at the same depth will combine to produce the larger combined area. More detail can be found on this algorithm in Rosselot[14].

4.3 Results and Discussion of the XH-map algorithm

The speed of the XH-map algorithm is remarkably fast, executing a single frame in 1.4 milliseconds on a Pentium 4 computer. The XH-map algorithm may be applicable to a variety of other problems in computer vision, and as a first step in wide class of other problems. For example, in trying to identify an object in space the first step would be a quick search using the X-H May algorithm to determine the size of the object matches the expected value.

REFERENCES

[1] Welch, G. and Bishop, G., "An Introduction to the Kalman Filter", TR95 041, University of North Carolina at Chapel Hill, Department of Computer Science, (1995).
[2] www.cs.unc.edu/~welch/kalman/.
[3] Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," Transaction of the ASME—Journal of Basic Engineering, 82(D), 35–45 (1960).
[4] Julier, S. J., Uhlmann, J. K., and Durrant-Whyte, H. F., "A new approach for filtering nonlinear systems," Proceedings of the American Control Conference, 3, 1628-1632, (1995).
[5] Simon, D., [Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches], John Wiley & Sons New Jersey, 448-450 (2006).
[6] Ibid., 132.
[7] Vince, J., [Essential Mathematics for Computer Graphics], Springer-Verlag London, (2001).
[8] www.min.uc.edu/robotics/research/UKalFilter.m/.
[9] www.ptgrey.com/.
[10] Point Grey Research, "Triclops StereoVision System Manual Version 3.1" (2003).
[11] Matlab and The MathWorks Inc, "Image Processing Toolbox," (2001).
[12] Intel Corp, "Integrated Performance Primitives for Intel®Architecture Reference Manual," Volume 2: Image and Video Processing, Volume 3: Small Matrices, (2003).
[13] Intel Inc., "The OpenCV Open Source Computer Vision Library," www.intel.com/research/mrl/research/opencv.
[14] Rosselot, D., " Processing real-time stereo video in disparity space for obstacle mapping," MS Thesis, University of Cincinnati, (2005).