MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Depth Perception Using a Trinocular Camera Setup and Sub-Pixel Image-Correlation Algorithm

Joshua Migdal

TR2000-20 December 2000

Abstract

Published in TR-2000, May 2000, 15 Pages abstract: The ability to perceive depth is common in animals, including humans. It is an ability we take for granted in our everyday lives. To perceive depth, our brains interpret the two slightly different images it receives from our eyes. Similarly, multiple images are needed for computer depth perception. Depth perception algorithms must determine where objects are in one picture in relation to the others. The disparity data is then used to calculate depth. Recognizing these objects is difficult. Using more than two images helps the detection process. We present a trinocular camera setup to take three images of a scene instead of two. The depth extractor works well alone, and it works better with the preprocessor and postprocessor. Taking three views of a scene is more helpful than two, because occlusions are reduced and the effects of shadows are reduced. The algorithm can use the standard two-image format for comparison. With standard test images the algorithm performs well.

REVISION HISTORY

1. First version, TR2000-20, May 19, 2000.

1.0 INTRODUCTION

1.1 Depth Perception in Nature

Human beings, along with other animals, possess the ability to estimate how far objects are away from them. This is because we have two eyes that are offset from each other. Each eye sees a slightly different view of the world, which can be demonstrated easily by holding a finger up close to your face and looking at it from at first one eye and then the other. This difference in the field of view of each eye provides disparity data which is interpreted somehow by the brain into a sensation of depth. Disparity is the slight shifting that occurs in the placement of objects in each eyes' field of view. The closer an object is to your eyes, the larger the disparity becomes. This is why a finger held up close to your face appears to "jump" significantly between the different views of your two eyes. Depth perception is done quickly and effortlessly by our brains and is taken for granted in our everyday lives.

1.2 Computer Vision: Depth Perception

Like human vision, computer vision algorithms require multiple views of a scene to extract that scene's depth information. To extract depth information from a set of images (called stereo pairs if there are two images in the set), the algorithms must match certain portions of one image to portions of another. The offset, or disparity, between the matched parts in the two images is inversely proportional to its depth in the scene. These disparities can be converted into absolute depths by the formula $z = \frac{sf}{d}$, where z is the distance from the cameras to the object in the scene in meters, s is the separation distance of the cameras in meters, f is the focal length of the camera's lenses in pixels, and d is the calculated disparity in pixels. Since the resolution of depth is directly proportional to the resolution of the images, the higher the resolution of the images, the more precise the depth information will be since the disparity information will become more precise as well. It is unnecessary for depth algorithms to calculate the absolute depth for a scene since relative depths using only disparity data can be mapped to absolute depths. Depth maps, which are usually grayscale images representing depth information in shades of gray (by convention, white is close, black is far), can be calculated using only disparity data.

1.3 Methods for Algorithms

There are two common methods for detecting parts of one image with parts of another and thus extracting depth information: area- and feature- based detection.

Area-based detection works by correlating one particular pixel in one image with its corresponding pixel in another image. Since one pixel in an image can be potentially matched to many pixels in another image, the pixels surrounding that one are used to help find the proper match. This image "patch" is then correlated with a similar "patch" in the corresponding stereo pair and the pixel's disparity data is thus calculated. For this reason, these algorithms are called correlation-based schemes. This process is repeated for each pixel in the image.

Feature-based detection works by first translating the raw pixel data into a set of features that can be recognized. This is done because it is assumed that a set of features is a more stable image property than raw pixel values. The algorithm then attempts to match the set of features with those features present in the corresponding stereo pair. Disparity data is then calculated based on these features, but only on these features. Pixels not present in any feature will not have a disparity value associated with them.

2.0 TRINOCULAR VISION SYSTEM

2.1 Introduction

A trinocular view was used to extract the depth information from an arbitrary scene using a correlation-based algorithm. A trinocular view consists of three separate views of the same scene, as opposed to a binocular view, which uses only two. A trinocular vision system has certain advantages over a binocular vision system that were exploited.

Since there are three separate views instead of two, you have more data against which to find a good match for a specific pixel. This helps because certain visual problems appear when obtaining separate views of the same scene. Occlusion is the biggest problem that faces a correlation-based method for determining depth. Occlusion is when an object or part of an object in the background is hidden from view by an object in the foreground. This occurs because objects in the foreground shift more than objects in the background in relation to the different camera views. An object or part of an object present in one view may not be present in another. With three camera views instead of just two, the problem of having an object occluded is lessened as it is less likely the object will be occluded in more than one offset view.

2.2 Setup

Only one camera was used to obtain the three separate images of the scene necessary to determine depth. The camera took a picture of the scene and was physically shifted before it took the next picture. Because the images were taken at different moments in time, depths could only be calculated for stationary scenes.

There were many reasons only one camera was used rather than three. The cameras interfaced with the PC via a parallel port, which has neither the number of pins nor the speed to control more than one camera at a time. The cameras themselves were low cost and their lenses often produced images such that they were unable to be calibrated accurately with each other. Using only one camera makes calibration unnecessary.

The camera was attached to a clamp which was mounted on a heavy metal plate used in laser laboratories. A 1-inch square grid was placed on the table below the metal plate so that its position could be known precisely. The camera took the rightmost view of the scene and the metal plate was moved one inch horizontally across the table. It took the center view and was again moved one inch to the left. It then took the leftmost view of the scene.

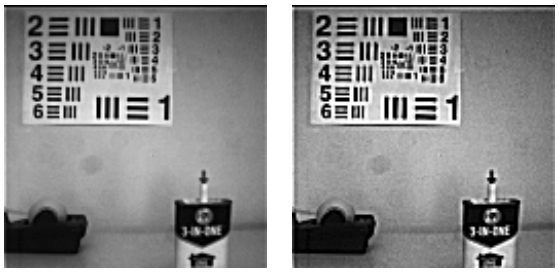
The camera's imaging sensor is a Mitsubishi CMOS imaging sensor capable of obtaining 128x128x8-bit grayscale images. This image size and color depth is ideal for such applications as robot vision and proximity detection. Since it is nowhere near the resolution of the human eye, the scenes that the camera took pictures of were small and relatively close by and uncluttered. Because the resolution is so small, its depth resolution is not great. So limiting its applicable range enhances its ability to determine precise depth, within that range.

2.3 The Depth Algorithm

The algorithm used to extract the depth information from the scene is a correlation-based algorithm developed by Joshua Migdal under the supervision of Dr. William Yerazunis for Mitsubishi Electric. The algorithm has three parts: a preprocessor, a depth extractor, and a postprocessor.

The preprocessor applies a sharpening filter to the images so that the edges stand out more. This works very well because edges are easy to identify in the different images and thus

can be used to obtain accurate depths. If more edges are brought out through a sharpening of the image, the pixel matches will be greater.



Preprocessing of a particular view

The depth extractor is the correlation-based algorithm itself. It uses the center image as the basis for calculating the pixel depths and generating the resulting depth map. The center image matches against the left and right views, but the left and right views do not match against each other. There are several parts that make up this algorithm.

The convolution window, which determines how large of an image patch will be used to determine the depth of a pixel, is a parameter given to the algorithm. There is an art to setting this parameter. If the patch size is too great you will sacrifice depth resolution. This is because pixel offsets become less important for the quality of a match the larger the patch size is. But if you make the patch size too small, then false matches occur more often as there is less data to base a good match on. A reasonable size that works well is a window of 7 pixels square, with the matching pixel the centermost one.

Another parameter is the disparity limit. It is useful to limit how far away the algorithm will look to find a match for a pixel in a particular view. To limit the range reduces false matches as the algorithm has less data to match against (so long as the disparity limit is sufficient enough to *find* the correct match!). It also makes the algorithm faster, which is important for practical uses. But this parameter is arbitrary: it depends upon the view. If we knew how much an object would shift beforehand, then we wouldn't need to calculate its depth; we would know it already. But this knowledge is precisely what we need to automatically choose a disparity limit. So in practice this should not be done.

There are techniques to limiting the number of pixels that need to be checked without having any advance knowledge of the scene. Because of the way the camera views are arranged, we know that any disparity will be along a horizontal scan line only (There is no vertical shifting of the images). Thus it is only necessary to check pixels on the same horizontal scan line in a shifted image as the original pixel in the center image. To limit further, we need only check to one side of the original pixel in a shifted image. Depending on which offset view the match is being attempted on, you need only look either to the right or to the left.

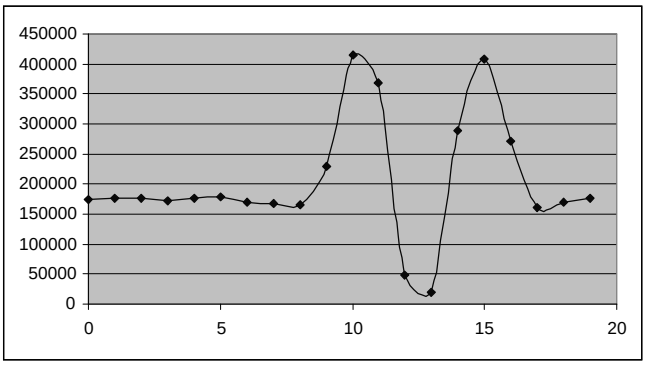

Left View


Center View

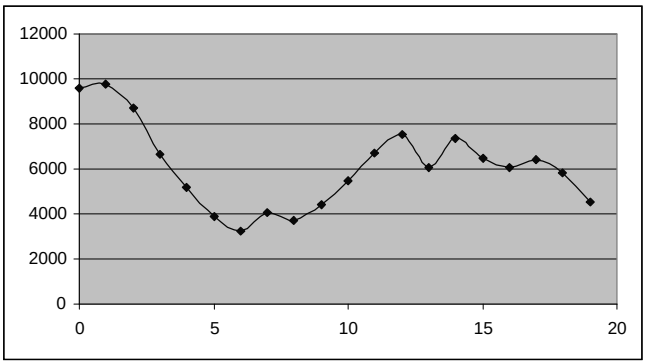

Right View

For example, suppose we were matching the pixel shown (indicated by the circle) above in the center image against the left and right views. In the left view, this pixel is offset to the left from the center image. Likewise, the same pixel in the right view is offset to the right. It is because of this that the views get their name. In reality, the left view is taken when the camera is in the rightmost position and the right view is taken when the camera is in the leftmost position.

To determine if a particular match along the horizontal scan line is good, a method known as the sum of squared differences is used. This method subtracts the raw intensity values of each pixel in the patch from the corresponding pixel in the offset image and squares it. Add up all the squared differences for the entire patch and you get one value called the sum of squared differences (SSD) for the offset pixel. The reason for squaring the differences is to weigh the differences in pixel values non-linearly. This gives near-misses in pixel values between center and offset view patches much greater weight than large misses in pixel value. The smaller the SSD value for an offset pixel (and for its convolving window), the better the match.

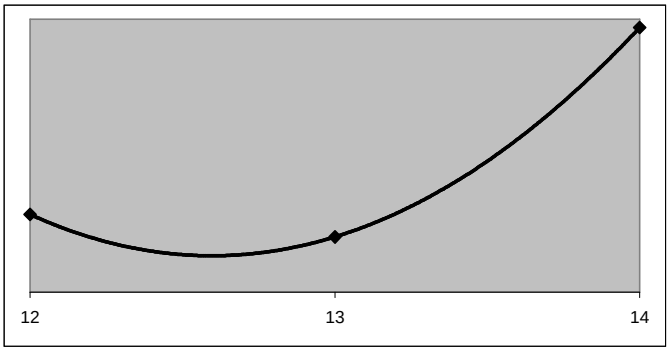


Good Match



Bad Match

This figure shows examples of a good match and a bad match along a horizontal scan line. The basic necessity for a good match is overwhelming evidence that a specific offset identifies the original pixel in an offset image. The good match has a very sharp decline and a very sharp incline around the smallest SSD value in the group. Coupled with this is the fact that no other local minimums have values anywhere near the absolute minimum. These are the primary features of a good match. A bad match has just the opposite. It has many local minimums near in value to the absolute minimum value of the group and no sharp declines or inclines around any of the minimum values, especially the absolute minimum value.



Fitted Parabola

By interpreting the SSD data, it is possible to obtain sub-pixel depth perception. This is achieved by fitting a parabola through the absolute minimum SSD data point and the points to either side of it. The actual depth will be the minimum of this fitted parabola, not the absolute minimum SSD data point. The formula to determine the sub-pixel depth is: $\Delta = \frac{c - a}{4b - 2a - 2c}$, where a is the pixel to the left of the absolute minimum SSD value, b is the absolute minimum SSD value, c is the pixel to the right of the absolute minimum SSD value, and Δ is the calculated offset from point b. The actual depth of the pixel would therefore be $(b+\Delta)$.

The postprocessor refines the depth map generated during the depth extraction. It does so by analyzing the SSD data for each pixel of the center image and assigning to that pixel a match quality. This match quality is based upon the qualities of a good match: a low absolute minimum, no ambiguous local minimums, and a steep fall before the minimum and a steep rise after. If a pixel has these qualities, it is a near certainty that a match was found. If it had a couple of these qualities, the match isn't as good but it isn't bad either. If it had none, then the match is uncertain. After assigning each pixel a match quality, it then refines the depth map by changing the depth values of pixels it is uncertain about. The postprocessor does this by performing a proximity search of pixels around the pixel in question looking for pixels with a better match quality than itself. It then averages the depths of all better matched pixels around that level in the proximity search. This has the effect of smoothing the depth map and removing discontinuities within the depth map.

3.0 RESULTS

3.0.1 Introduction

The results consist of several different images. The results show the images used in the separate views. For two scenes, there is a left, right, and center view taken by the cameras. For the others, there are only two views, right and left. These other data are standard stereo pairs used to by many depth extraction algorithms and are useful in comparing against the different algorithms. In these cases, the right view is used to generate the depth map.

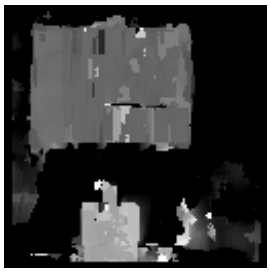
There are two depth maps: the original depth map and the recalculated depth map. The recalculated depth map is the final result, after postprocessing. There is also a match map, which shows the strength of the match associated with each pixel's depth (the lighter, the better). All of these maps are rescaled so that the variations in color, representing depth and match strength, are more apparent.

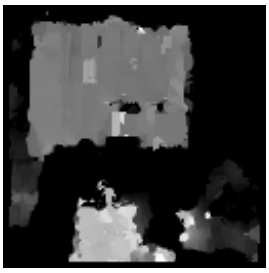
3.1 Single Object Scene, No Preprocessing

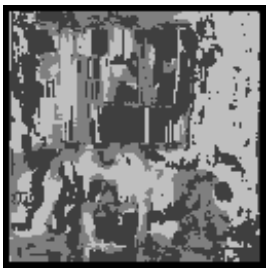

Left View


Center View


Right View


Depth Map


Recalculated Depth Map


Match Map

3.1.1 Scene Description

This scene consists of an oil can on a table in the foreground, and a calibration sheet taped to the wall in the background.

3.1.2 Algorithm Analysis

The depth map shows clearly that the oil can was found to be in the foreground and the calibration sheet in the background. The recalculated depth map removes some white from the tip of the oil can in the depth map and smoothes the calibration sheet, giving it a more uniform color (depth). The match map is surprising since it describes the quality of match associated with most of the oil can to be poor. This did not seem to affect the quality of the depth map, however.

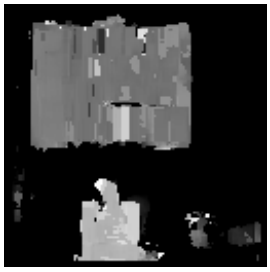
There are trouble spots in the resulting depth map. The area to the lower-left is filled with bad data. This is due to the lack of good distinguishing features that a correlation-based algorithm needs to find a suitable match. Another problem is that the algorithm found the wall to be far behind the calibration sheet, which is on the wall. This is due to sensor noise and the fact that there are no distinguishing features on the wall. This allows the sensor noise to prevail over features present in the image and become the dominant matching feature. Since the sensor noise does not shift with the different views, the algorithm assumes there is no shifting. Worse yet, it found that this sensor noise was of the very best quality match!

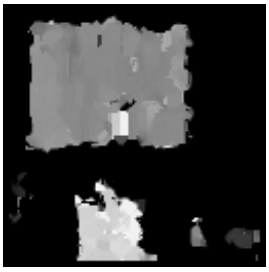
3.2 Single Object Scene, Preprocessing

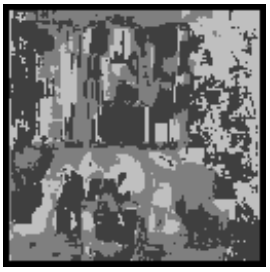

Left View


Center View


Right View


Depth Map


Recalculated Depth Map


Match Map

3.2.1 Scene Description

This scene is the same as section 3.1, but with the sharpening filter applied.

3.2.2 Algorithm Analysis

The performance is much better on the preprocessed images. The ambiguities in the lower-right corner of the depth map are reduced, the continuity of the depths for the calibration sheet and oil can are much improved. The recalculated depth map shows a much greater level of refinement than for section 3.1.

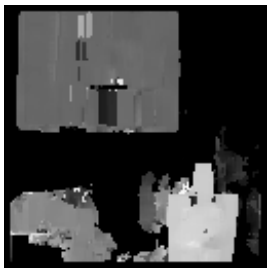
For better of for worse, the preprocessing made the sensor noise even more blatant, so that ambiguities in the wall present in section 3.1 are reduced or eliminated here.

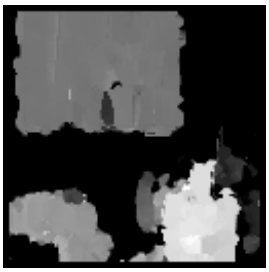
3.3 Multiple Object Scene, No Preprocessing

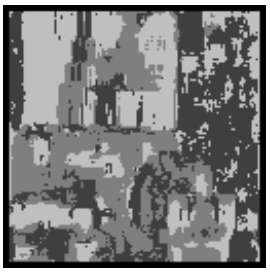

Left View


Center View


Right View


Depth Map


Recalculated Depth Map


Match Map

3.3.1 Scene Description

This scene consists of three objects: an oil can in the foreground, a tape roller midway between the wall and the oil can, and a calibration sheet taped to the wall.

3.3.2 Algorithm Analysis

The algorithm performed well. It found the oil can, the tape roller, and the calibration sheet, each at their proper depths. It reported the depths of these objects with much uniformity, especially noticeable in the recalculated depth map. The match map shows that the algorithm was much more sure of the calibration sheet and oil can than it was in both sections 3.1 and 3.2.

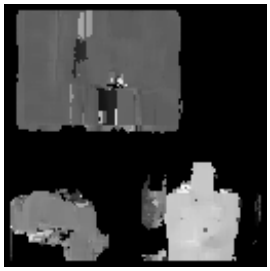
The algorithm again had trouble with the sensor noise in regions of the images sparse with discriminating image data (i.e. the wall and table areas). What seems like a “melting” of the tape roller in the depth maps is actually a successful match of the table. The reason the algorithm picked this area up is because there are shadows present for the algorithm to match against. The algorithm also picked up artifacts of the oil can on the left hand side. This is most likely due to the shadow of the oil can cast on the wall behind it. Shadows, as well as specular highlights and reflections, vary significantly between views of the same scene and are not, in general, good features to match on.

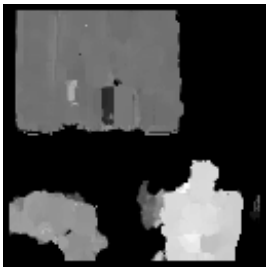
3.4 Multiple Object Scene, Preprocessing

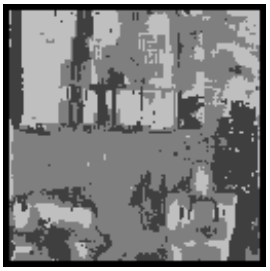

Left View


Center View


Right View


Depth Map


Recalculated Depth Map


Match Map

3.4.1 Scene Description

This scene is the same as section 3.3, but with the sharpening filter applied.

3.4.2 Algorithm Analysis

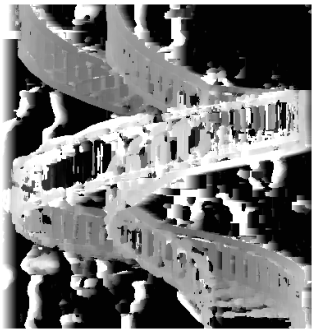
Predictably, the algorithm performed better on the preprocessed images. Most of the artifacts on the depth map due to shadows cast by the oil can are resolved correctly. The calibration sheet, oil can, and tape roller have very uniform colors (depths) and they were each found in their proper depths.

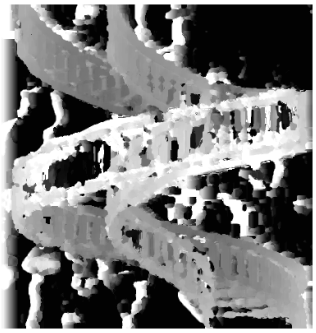
The preprocessing enhanced the sensor noise as well, yielding very good “matches” to the wall and desk areas as well. Unfortunately the sensor noise is not shifted at all in the different views so it appears that the desk and wall are very far in the background. Ideally, the calibration sheet should not be distinguishable from the wall in the depth map, as they should both appear at the same depth.

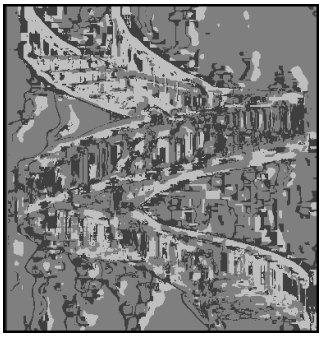
3.5 The Staircase


Left View


Right View


Depth Map


Recalculated Depth Map


Match Map

3.5.1 Scene Description

This is a scene of a spiral staircase, rendered in POV-RAY. Because it is rendered, it makes a perfect stereo pair. There is no camera noise, lens distortion, calibration issues, etc. This makes the scene great for testing a best-case scenario, but will typically not show real-world results.

3.5.2 Algorithm Analysis

The algorithm performed very well under these ideal conditions. The depth map has a nice gradient representing the staircase which circles nearer and further away from the “cameras” as it spirals up. The background, for the most part, is completely black, representing the fact that the background does not shift between images.

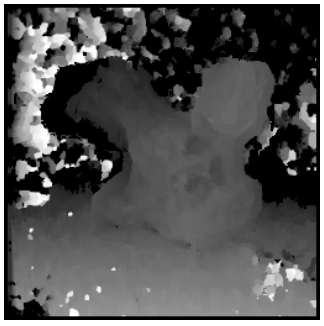
“White strings” appear through the black background of the depth map, however. These show up sporadically through the background, probably because there is little distinguishing features at those parts in the background. The “white string” areas are complemented nicely with areas of low match quality in the match map. This means that the algorithm caught that they were bad matches, but could not do anything with them. It is difficult to see in these shrunken pictures, but in the recalculated depth map, the “white strings” are dotted with black dots, representing those areas where the postprocessor attempted to fill in the bad data with better data.

3.6 Car Part

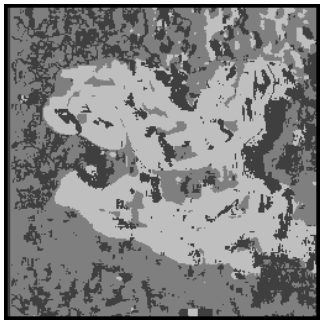
Left View

Right View

Depth Map



Recalculated Depth Map



Match Map

3.6.1 Scene Description

This is a scene of a Renault automobile part. This is a real scene, taken with cameras; it is not rendered. It is not quite a natural scene, however, since it was set up under ideal conditions.

3.6.2 Algorithm Analysis

The car part was found nicely, as was the table it is sitting on. The depth map shows a very nice gradient of color (depth) representing the table sloping off into the background. The car part has well-defined regions in the depth map, representing its odd shape and contour. The match map shows that the algorithm was very sure of most of the car part and some of the surrounding area. The algorithm was reasonably sure of the table area.

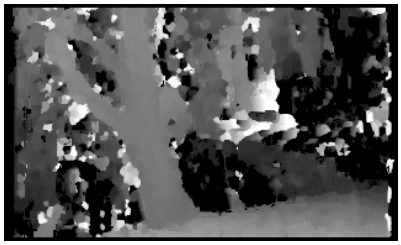
Predictably, the algorithm had trouble with the areas of little definition. The areas in the depth map that are discontinuous in color (depth) are the very areas in the match map where the algorithm showed little match strength. The postprocessor adds depth information to some of the pixels with little match strength, but not enough to significantly enhance the depth map.

3.7 Tree, No Preprocessing

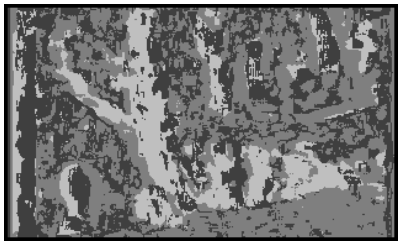
Left View

Right View

Depth Map



Recalculated Depth Map



Match Map

3.7.1 Scene Description

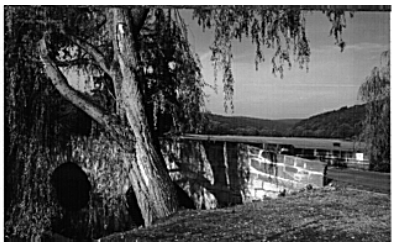
This is a natural scene of a tree, a ledge, and a small brick wall. There are mountains present in the background. This is a very dense image, with lots of differing depths. The resolution of these images is not such that the depths could be extracted with as much detail as could be inferred simply by looking at the images. This stereo pair makes for a good real-world test.

3.7.2 Algorithm Analysis

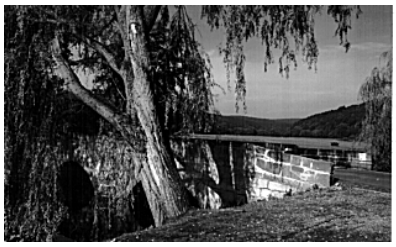
The algorithm found the tree and its branches nicely. The color (depth) in the depth map is very smooth around the tree area. It also accurately found the ledge that the tree is growing from. There is a nice gradient that marks the ledge's fading into the background. The recalculated depth map does a great job in removing the discontinuities within the gradient of the ledge.

Although the algorithm found the objects in the foreground accurately, it had trouble finding the object far in the background. The background appears with numerous white patches in it. Shadows underneath the tree confused the algorithm as well. The dark spots caused by shadows have no spatial continuity between the different views of the scene and therefore are not suitable for matching. The algorithm does not know this, of course, and gets confused.

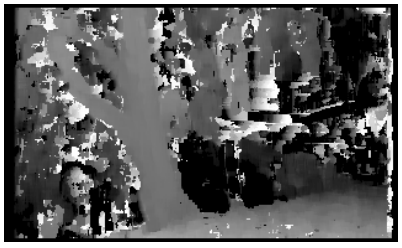
3.8 Tree, Preprocessing



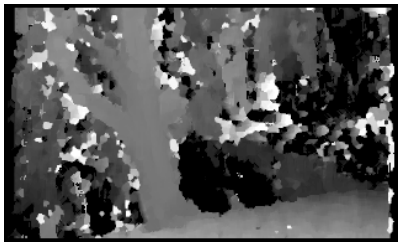
Left View



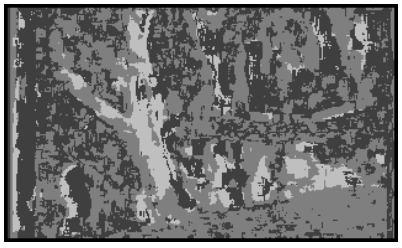
Right View



Depth Map



Recalculated Depth Map



Match Map

3.8.1 Scene Description

This scene is the same as section 3.7, but with the sharpening filter applied.

3.8.2 Algorithm Analysis

The recalculated depth map has a larger role in this scene than it did in any other. It performed very well to smooth out the inconsistencies in color (depth) present in the original depth map near the sky and branches of the tree. It removed nearly all of the inconsistencies within the gradient representing the ledge. The algorithm found the tree and ledge well, as it did in the unfiltered scene.

As was described in section 3.7, the algorithm found the foreground objects with more accuracy than the background objects. The match map and the position of the poor quality matches correlates well with the false matches present in the depth map.

4.0 CONCLUSION

4.1 Interpretation of Results

The tests performed show clearly that the preprocessing step aids the depth extraction process. It does so by enhancing the edges and creating more discriminating features to match against. This cuts back on false matches and generates more good matches.

The postprocessing step is also helpful in creating the final depth map. It removes certain discontinuities within well-matched areas. The postprocessor has great potential to help generate accurate and continuous depth maps. The results show that the postprocessor makes valid choices for pixels with bad match qualities, though the results also show that the postprocessor needs to do more work. It does not fix enough bad match-quality pixels and so large areas of discontinuity still exist.

The depth extractor performs predictably. It gets caught up on shadows and areas of low detail or areas of low discriminating features. Taking multiple views of the same scene helps to cut down on these problems by giving the algorithm more features to match against. This is especially useful for shadows and reflections, as it is unlikely that the same shadow will be cast at the same spot in multiple views. So to have multiple views gives the algorithm more chances to make a good match based on actual data, not shadows or highlights.

4.2 The Next Step

The next logical step will be to improve both the depth extractor and the postprocessor. A study of how the brain interprets depth could reveal new ways for extracting depth. People who see only one picture have a sense already of which objects are in the background and which are in the foreground. We also have the ability to understand perceptive. We can tell if an object is fading towards the background or coming out towards the camera. Since these do not require multiple views but are left more to human interpretation, it is not so much a mathematical problem but a psychological one. To implement this common sense algorithmically would be challenging, but perhaps necessary. The brain also has the ability to fuse the two separate images collected by our eyes into a cyclopean view. This means that the brain itself, without conscious interpretation, has the ability to fill in data left out from either the left or right eye by occlusion and other phenomena.

Another way to perhaps enhance the abilities of the depth extractor is to change the way the test images are taken. Right now, the three views are parallel to each other. This is not the way our eyes see the world, however. Both of our eyes focus on one particular spot at a time. This is why our eyes go “cross-eyed” when trying to see a finger held up close to our face. So, instead of having the cameras parallel to each other it may be beneficial to position then so that they create a focal point in space.

The postprocessor could also be improved by making it more robust. It should fix more trouble spots in the depth map. A way to make it more robust is to have the proximity search go down more levels in its search to find good quality matches. Also, it could be made to refine the depth map iteratively, going over the image in multiple passes fixing more and more problem areas by using the data collected during previous passes.