REAL TIME OBSTACLE DEPTH PERCEPTION USING STEREO VISION

By

TIANYU GU

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2014

To my mother, Hongmei Zhou, and my father, Siqi Gu

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to Dr. Carl Crane, my supervisory committee chair, for his generous support, encouragement and guidance. I would also like to thank Dr. John Schueller for serving on my supervisory committee. It was a great honor to have both of them on my committee and give me lectures in robotics and computer control.

I would also like to thank Dr. Ryan Chilton for his guidance in stereo vision. I thank Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz and Dr. Josh Weaver for providing me the opportunity to learn robotics through Intelligent Machine Design Laboratory course. Each of them carries a special meaning for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

REAL TIME OBSTACLE DEPTH PERCEPTION USING STEREO VISION

By

Tianyu Gu

May 2014

Chair: Carl D. Crane, III
Major: Mechanical Engineering

Stereo vision is often used to recover depth information by mimicking how human eyes perceive the distance of an object. It has many applications like 3D reconstruction, robot grasping and robot navigation, etc. While the development of stereo correspondence algorithms have been active research topic over the past few decades, very little attention has been paid to their implementation on real-time mobile robots. This study builds a fast and inexpensive stereo vision system for real-time purposes. A local dense stereo correspondence algorithm called block matching was implemented on our system using OpenCV. Experiments showed that our system is capable of perceiving depth of moving objects at a frame rate of 30 Hz. To further simplify the disparity map results, a foreground extracting technique based on Otsu's method was tried prior to stereo correspondence. Results showed that this technique only worked in an indoor environment with controlled ambient lighting. Further developments on hardware and software are suggested to improve the robustness of a segmentation-stereo combined approach.

CHAPTER 1
INTRODUCTION

Stereo Vision at a Glimpse

Stereo vision, just like human eyes, infers distance from two images taken from different views. It is based upon the fact that the same object would appear at slightly different locations in the left and right eye of a human, which is recognized as a disparity. People will have trouble in walking or grasping things if they close one of their eyes. Similarly, a robot can hardly extract depth information from the environment only with a single camera. Therefore, the research of stereo vision systems is of key importance.

Binocular stereo vision uses a pair of frontal parallel cameras to infer distance by calculating the disparity of pixel in different image planes, as shown in Figure 1-1. A scene object P is viewed by both left and right camera simultaneously, and the position of P on each image is denoted as x^l and x^r respectively. Let's assume that the focal lengths f of both cameras are the same, and the images are undistorted and row-aligned. The distance of object point P can then be easily calculated from the similar triangles Px^lx^r and PO_lO_r , which yields

$$\frac{T+x_l-x_r}{Z-f} = \frac{T}{Z} \tag{1-1}$$

solving for Z gives

$$Z = \frac{fT}{x_r-x_l} \tag{1-2}$$

We can see that depth is inversely proportional to the disparity. This is true when you look at same object using different eyes: things that are closer will shift more than those farther away.

Stereo Correspondence

One of the major problems of stereo vision is stereo correspondence. An image is comprised of hundreds of thousands of pixels. Each pixel is a result of a scene point being

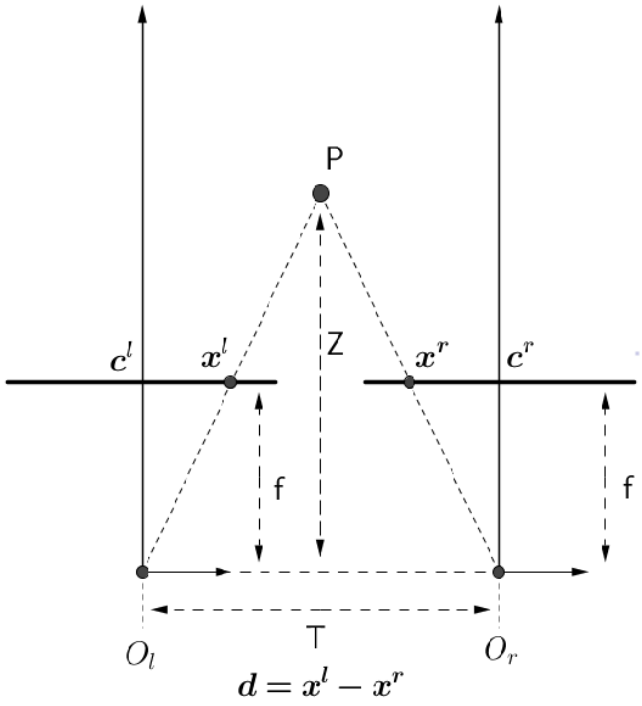


Figure 1-1. Depth Z can be recovered from similar triangles. O_l and O_r are centers of projection; Optical axes of both camera extends from O_l , O_r and go through image planes.

projected onto the image plane. Before we can apply equation 1-2 to solve for depth, a certain technique has to be used to find the corresponding pixel in both images (left and right). It is conventional to declare the left image as the reference image while regarding the right image as the target image. For a given pixel in the reference image, once its corresponding point is located in the target image, the disparity for that pixel is derived. A complete stereo correspondence algorithm will compute disparity for each pixel that appeared on both images. The result of stereo correspondence is a dense disparity map as shown in Figure 1-2.

A large number of stereo correspondence algorithms have been developed, and they can be broadly categorized into two groups: window-based (local) cost matching and global method that solves a minimization problem. Details of these algorithms and their comparison will be reviewed in Chapter 2. While these algorithms differ from each other, most of them use the same

epipolar constraint to reduce their search space from 2D to a single scan line, which significantly reduces time complexity.

Figure 1-2. *Tsukuba stereo dataset*. Source: Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1), 7–42. Reprinted with permission from Middlebury Computer Vision Pages, http://vision.middlebury.edu/stereo/data/ (Dec 7, 2013). Left to right: left image, right image, disparity ground truth.

Epipolar Geometry

Epipolar geometry is introduced in the interest of narrowing down the search space to a one dimensional epipolar line. In practical implementation, cameras can hardly be perfectly aligned so they don't have the frontal parallel configuration as shown in Figure1. Instead, cameras are usually tilted a little bit with respect to each other, thus giving rise to epipoles [Hartley 03]. An epipole e_l (e_r) is defined as the projection of the center of projection O_r (O_r) onto the image plane $\prod l$ ($\prod r$). An epipolar plane is the one that contains all three points: physical point P, e_l , e_r. Finally, an epipolar line is formed if we project line PO_r onto, for example, the left image plane $\prod l$. Once we are given a point p_l in one imager, the search for p_r in the other imager will be along the epipolar line $e_r p_r$. This is known as epipolar constraint.

Problem Statement

The development of a real-time stereo vision system on a mobile robot poses many problems such as computation speed, camera synchronization, and 3D reconstruction. A moving

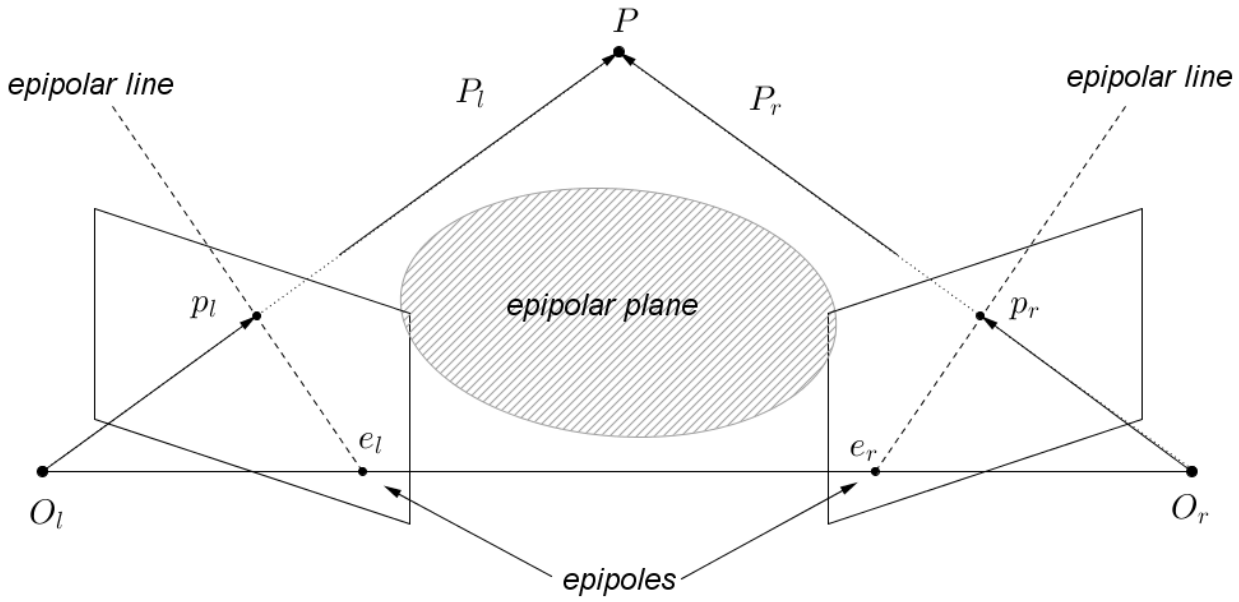


Figure 1-3. Epipolar geometry of stereo vision system

robot requires immediate depth information upon capturing an image. Likewise, if the camera system is moving with the robot or it is capturing a moving object, both of the cameras have to be synchronized in order to perform stereo matching on the same pixels. Moreover, a dense disparity map usually contains background noise that makes it difficult for obstacle-specific depth calculation. In this research, we propose a low-cost stereo vision system that mainly solves the following two problems:

1) Achieve fast stereo correspondence at a frame-rate of 30fps using webcams and a laptop computer. This requires that a complete stereo processing cycle does not exceed 33ms. In this research, computation speed outweighs the accuracy of the disparity map. Both indoor and outdoor scenes will be chosen to test the system's capability to perceive depth in a real-time manner.

2) Simplify the disparity map using a foreground extraction technique prior to stereo correspondence. The original disparity map usually contains background depth information and

noise that do not contribute to depth perception. The implementation of foreground extraction in an outdoor environment requires expensive auto-iris cameras. Since inexpensive webcams are being used here, we only examine this technique in an indoor environment.

CHAPTER 2
STATE OF THE ART

Overview of Depth Perception Technologies

There are many approaches to perceive depth of an object, among which are structured lights, laser ranging, and stereo vision [Se06]. The structured light method projects a known pattern onto the target and then infers depth from the deformation of that pattern. This technique is used in many indoor applications such as Microsoft Kinect. While structured light is ideal for low light environments and low textured scenes, its relatively short range (0.8m – 3.5 m) prevents it from being used in outdoor applications like robot navigation. The laser ranging method, also known as TOF (Time of Flight) method, computes the distance of an object by calculating the time it takes for the light to have a round-trip between the light source and the object. Examples are the Velodyne 3D LIDAR that is used on many autonomous robots such as Google's driverless car. The laser scanning technique has many advantages like long ranging distance, high accuracy, and efficiency. However a laser scanner might suffer noisy readings in bad weather conditions, since rain drops or snowflakes would disturb the measurement. Finally a laser scanner is usually very bulky and cost prohibitive. Stereo vision is a completely passive sensing technique. A binocular stereo vision system has two parallel cameras taking pictures of the same scene from slightly different positions. It then calculates the depth of each scene point using simple triangulation geometry and therefore depth of the whole scene could be recovered. The stereo vision system is inexpensive to build and efficient in depth recovering. However stereo vision is computationally expensive and sensitive to ambient light and scene texture.

Previous Development of Stereo Vision System at UF

The Center for Intelligent Machines and Robotics (CIMAR) at University of Florida is a research center for autonomous vehicles and intelligent robotics. It has previously developed several stereo vision systems for autonomous ground vehicles.

NaviGator Stereo Vision System

Maryum F. Ahmed [Ahmed06] designed an outdoor stereo vision system assisting the navigation of CIMAR's NaviGator ground vehicle. The system produced a traversability grid that can be fitted into CIMAR's Smart Sensor Architecture.

Her system used s-video cameras with auto-iris lenses and a four frame grabber PCI card to perform analog to digital signal conversion (Figure 2-1). The SRI stereo vision library was used to perform stereo calibration, rectification, and 3D data calculation. The 3D points were then fitted into a plane that was used to find the slope of the terrain in the traversability grid.

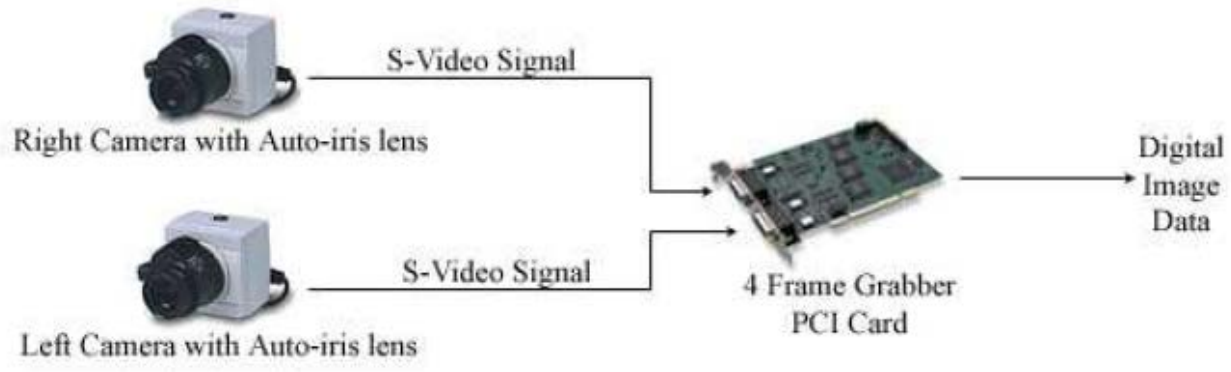


Figure 2-1. Hardware interface of NaviGator stereo vision system. Source: Ahmed, M. F. (2006). "Development of a Stereo Vision system for Outdoor Mobile Robots," M.S. thesis, University of Florida.

Tests were performed to find the best stereo correspondence parameters such as pixel search range, horoptor offset, correlation window size, confidence threshold, and uniqueness

threshold. Tests further showed that the auto-iris camera system performed better than the manual-iris camera in terms of their outdoor stereo processing result.

Videre Design Stereo Vision System

CIMAR has previously used a Videre Design camera to build two stereo vision systems. One was the Mega-D Wide Baseline and the other was the Mega-DCS Variable Baseline. The Wide Baseline stereo rig has two cameras at a wider, fixed position with respect to each other. The wide baseline enables the system to resolve more depth. On the other hand, the Variable Baseline stereo rig has two cameras that can be moved with respect to each other. The variable baseline makes the system more flexible in resolving different depths. However the cameras have to be re-calibrated after each move. The Mega-D Wide Baseline stereo rig is shown in Figure 2-2. They are using IEEE 1394 Firewire to interface with the computer.

Figure 2-2. Videre Design Mega-D wide baseline stereo camera rig with LIDAR laser rangefinder. Source: Ahmed, M. F. (2006). "Development of a Stereo Vision system for Outdoor Mobile Robots," M.S. thesis, University of Florida.

For the software, both systems used the SRI Small Vision System library that is capable

of adding stereo processing to user written applications. The library contains stereo rectification and calibration algorithms that are ready to use. In addition, Videre comes with an interface with the SVS, which makes the image capture and software implementation much easier.

Taxonomy of Stereo Correspondence Algorithm

A number of algorithms have been developed for stereo correspondence over the past few decades. They could be broadly categorized as sparse (feature-based) stereo correspondence and dense stereo correspondence [Szeliski 11]. In this section we will briefly review sparse stereo correspondence and elaborate more on the dense correspondence that has been actively researched nowadays.

Sparse Stereo Correspondence

Sparse correspondence was the early technique for stereo matching. It tries to find the matching features such as an edge or corner between the left and right images. The motivation for a feature-based method was due to a limited computation resource availability and a desire to find matches with high certainty. The early sparse correspondence methods extract a couple of potentially corresponding features in the reference image using either edge detector or interest operator, and then the target image is searched for a correlation using a patch-based metric [Marr&Poggio 79]. Later improvements tried to extract features first and then use them as "seeds" to grow additional matches between both images [Zhang & Shan 00]. The sparse correspondence method is becoming less popular due to the rapidly increasing computation power and growing need for dense disparity information in applications like image rendering and 3D reconstruction.

Dense Stereo Correspondence

The development of the dense stereo correspondence algorithm has remained active over years. The dense method can be categorized as local (window-based) and global optimization

methods. While some of them excel in the quality and detail of the disparity map, others are less computationally expensive and therefore much faster. Scharstein [Scharstein 02] has summarized a variety of dense correspondence and their performance as well. In his paper he presented a general procedure for dense methods regardless of their specific technique:

1) Matching Cost Calculation
2) Cost Summation
3) Disparity Computation
4) Disparity Refinement

Generally a local method might perform all of these four steps while a global optimization supplements local methods by making explicit surface continuity assumption.

Local (Window-based) Method – The local method searches for matching pixels according to the intensity of pixels only within a finite sized window. It sums up all the matching cost of the pixels within that window and assigns a disparity to the one with the lowest matching cost function. This is also known as the Winner-Take-All principle. Commonly used matching cost functions are Sum of Absolute Differences (SAD) [Kanade 94], Sum of Squared Differences (SSD) [Hannah 74] and Normalized Cross-Correlation (NCC) [Hannah 74]. However, since the local method only makes an implicit smoothness assumption, it will inevitably fail at disparity discontinuities, leading to "flattened" boundaries. Several algorithms have been developed to mitigate this effect, explicit occlusion detection [Egnal 02], and multiple windowing [Fusiello 97], to name two.

Global Optimization Method – The global method solves for the disparity d that minimizes a global energy function that includes a smoothness term. The global energy function is expressed as:

$$\mathrm{E(d)} = E_d(d) + \lambda E_s(d) \tag{2-1}$$

Explanations for each of the energy terms are below:

$E_d(d)$ - A measure of how similar the pixel at disparity d is to the reference. This is a similar scheme to the matching cost in the local method. This energy term can be expressed as:

$$\mathrm{E_d(d)} = \sum_{(x,y)} \mathrm{C(x, y, d(x, y))} \tag{2-2}$$

where C is the matching cost.

$E_s(d)$ – The smoothness term penalizes any disparity discontinuities. It can take either of the following two forms:

$$\mathrm{E_s(d)} = \sum_{(x,y)} \rho\big(\mathrm{d(x, y) - d(x + 1, y)}\big) + \rho\big(\mathrm{d(x, y) - d(x, y + 1)}\big) \tag{2-3}$$

$$\mathrm{E_s(d)} = \rho_d\big(\mathrm{d(x, y) - d(x + 1, y)}\big) \times \rho_I(\|\mathrm{I(x, y) - I(x + 1, y)}\|) \tag{2-4}$$

Where

ρ, ρ_d – Monotonically increasing function of disparity difference.

ρ_I - Monotonically decreasing function of intensity difference.

For the first form, the bigger the difference in disparity, the more dominant the smoothness term will be. Thus the minimization framework will penalize more on huge disparity differences. However, the first form assumes a smooth disparity everywhere and may get some poor results at object boundaries. To avoid this problem, Gamble [Gamble & Poggio 87] came up with the second form which actually encourages disparity discontinuities at image edges.

A number of algorithms have been developed to solve for disparity under a global energy minimization framework. Conventional approaches include simulated annealing [Geman & Geman 84], regularization and Markov Random Fields [Blake & Zisserman 87]. Later improvements include graph cuts [Roy & Cox 98] and belief propagation [Sun et al. 03] that are more efficient in their implementations.

CHAPTER 3
HARDWARE

Stereo Cameras

The camera used in this thesis is Sony's PlayStation3 Eye (Figure 3-1). It is a gaming webcam capable of capturing standard video with frame rates of 60 Hz at a 640×480 pixel resolution, and 120 Hz at 320×240 pixels. Compared to other generic webcams, the PS3 eye has a larger CMOS sensor that makes it ideal for low-light environments. It also has a low-latency USB performance which is desirable for the stereo vision application. The PS3 Eye comes with a two-setting adjustable fixed focus zoom lens. One is a 56° field of view for close-up framing and the other is a 75° field of view for normal framing. Above all, it is cheap and is built for computer vision applications with community-developed drivers and SDK.

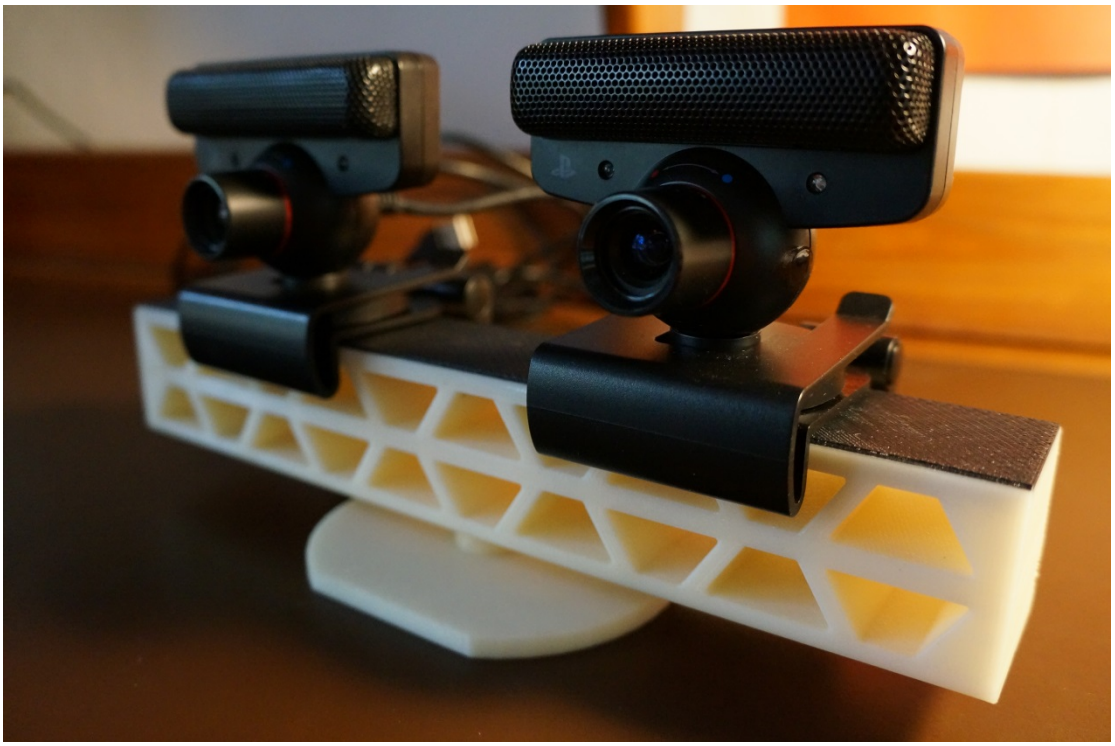


Figure 3-1. PS3 Eye webcams with a rapid prototyped base. Photographer: Tianyu Gu. Taken on Jan 14th, 2014 at Gainesville, FL.

Camera Synchronization and Image Resolution

One of the most important factors for a stereo vision system on a mobile robot is synchronization of the cameras. Suppose the robot is moving at a speed of 10m/s with the camera frame rate being 30Hz. Each frame of the camera is a snapshot of the scene every 0.33 m. If the synchronization is off by 10ms, then the two cameras are actually off by 0.1 m of the scene. This error is unacceptable during the process of stereo correspondence.

The PS3 Eye transfers raw image data to the host computer via high speed USB2.0. Once the stereo process begins, both cameras start to capture images at a certain frame rate and store raw images into a data buffer. Meanwhile a loop function is written to grab frames from both cameras simultaneously. To achieve optimal synchronization, image resolution and frame rate are carefully selected so that the total data volume does not exceed USB bandwidth. USB2.0 protocol allows a maximum of 60MB/s transfer rate. While in reality it could only get 40MB/s due to USB communication overheads. If we are transferring a pair of 320 x 240 24 bit RGB image captured at 30 fps, then the total bandwidth *B* would be:

$$B = 320 \times 240 \times 30 \times 3 \times 2 = 13{,}824{,}000 = 13.824\ MB/s \quad (3\text{-}1)$$

which is about 35% of the maximum USB2.0 bandwidth.

Similarly if we capture images at a frame rate of 60, then this would double the total bandwidth, which is about 70% of the maximum bandwidth. However, if the image resolution is 640 x 480, then the total would become:

$$B = 640 \times 480 \times 30 \times 3 \times 2 = 55{,}296{,}000 = 55.296\ MB/s \quad (3\text{-}2)$$

which exceeded 40MB/s. Thus in stereo vision applications using USB communication, we should limit our image resolution to 320 x 240 to achieve maximal synchronization.

Stereo Rig

The stereo rig is designed to have variable baselines. PS3 Eye webcams are tightly clamped to a plastic base that is rapid prototyped (Figure 3-1). Each camera's location can be easily adjusted along the base. Since there is no quantitative description of how much the baseline (distance between the optical axes of both cameras) should be, such a stereo rig allows us to adjust the baseline until a desirable effect is achieved. In general, for applications where the camera needs to see further (e.g. navigation), we might want to increase the baseline so as to gain more depth resolution. If the camera is expected to see close-up things (e.g. obstacle avoidance), then a shorter baseline is ideal since it increases the overlap between the two cameras.

Test Computer

The computer used is a Dell laptop with Intel Core i5 processor with a clock speed of 2.50 GHz with 6GB memory. It is running on a 64-bit Windows 7 Operating System. Both cameras are connected to the computer via USB 2.0.

CHAPTER 4
SOFTWARE

To estimate the depth of the scene we first need to solve the stereo correspondence problem. This chapter will introduce a fast stereo correspondence method called block matching. Since our system is designed to gear toward obstacle depth perception, a segmentation-based foreground extraction technique is further examined as the pre-processing step for stereo correspondence.

Stereo Correspondence

In the previous chapter we have introduced several stereo correspondence algorithms. In our research a local dense stereo correspondence method called block matching is chosen due to its fast implementation. Intel's Open source Computer Vision library (OpenCV) is selected to assist the research since it not only supports synchronized capture of stereo cameras but also provides many fast API for image processing. OpenCV comes with several stereo correspondence algorithms, among which stereo Block Matching (BM) and Graph Cuts (GC) are most frequently used. Since Graph Cuts turns out to be very computationally expensive despite its fine quality, this algorithm is not suitable for applications in real-time. The Block Matching algorithm is a local stereo correlation method that was first developed by Kurt Konolige [Konolige 97]. Although it produces a less accurate disparity map, it is very fast and thus is ideal for real-time purposes. Since the detection of obstacles on robots requires more speed than accuracy, we choose the Block Matching algorithm in this research.

If both of the cameras are calibrated and images are rectified (calibration and rectification will be carried out in the following chapter), then the left and right images are row-aligned and the epipolar geometry could be applied to searching for correspondence. There are three steps for the block matching algorithm [Bradski & Kaehler 08]:

1) Apply a pre-filter to normalize the image brightness
2) Search for stereo correspondence along the epipolar line using SAD window
3) Apply a post-filter to reject any bad correspondence

Pre-filtering

Stereo correspondence could be disturbed by lighting differences, illumination, and perspective. A local operator is applied to original image pairs to compensate for the disturbances. Basically each pixel's intensity is normalized by the average intensity within a local window around it. If we denote the intensity of the center pixel under a local window as I_c, then its value is replaced by:

$$I_c = \min\left[\max\left(I_c - \bar{I}, I_{cap}\right), I_{cap}\right] \quad (4\text{-}1)$$

Where $\bar{I}$ is the average value in the window and I_{cap} is a positive numeric limit whose default value is 30.

Matching Process

The matching process is carried out with a sliding SAD window, which can be expressed by the following equation:

$$\text{SAD Matching Cost} = \sum_{i=-N}^{i=N} \sum_{j=-N}^{j=N} \left| Il_{i+x_0, j+y_0} - Ir_{i+x_0+d, j+y_0} \right| \quad (4\text{-}2)$$

Where Il, Ir represent the pixel intensity in the left and right image respectively.

By convention, the left image is treated as the reference while we search along the corresponding row in the right image for the match. For each pixel (x_0, y_0) in the left image, there will be a support window around it. The support window has the same size as the SAD window and records intensity value of all pixels under it. Meanwhile there will be a scan line in the right image corresponding to(x_0, y_0). For rectified images the scan line is just the epipolar line. The matching search will start at Min Disparity, which defaults to zero. Then the search will

be carried out over a certain number of pixels that is controlled by Number of Disparities. For each pixel to be searched in the right image, there will be a correlation window around it. The correlation window has exactly the same size as the support window in the left image. Then, each correlation window will be correlated with the support window using the SAD (Sum of Absolute Differences) operation. Finally, the one with the minimum matching cost will be chosen as the pixel that forms the disparity, and this method is also known as Winner Take All (WTA). The matching search could be illustrated as Figure 4-1.

Post-filtering

The disparity map suffers from false matches that must be filtered. A couple of post-filters could be applied in the block matching algorithm. The first filter to prevent false matches is called uniquenessRatio which ensures that the candidate correspondence pixel has a distinctively small SAD value:

$$\mathrm{SAD}(\mathrm{d}^*) \times \left(1 + \frac{\mathrm{uniquenessRatio}}{100}\right) < \mathrm{SAD}(\mathrm{d}) \qquad (4\text{-}3)$$

where d^* is a candidate disparity and d denotes any disparity that is not equal to $d^* \pm 1$.

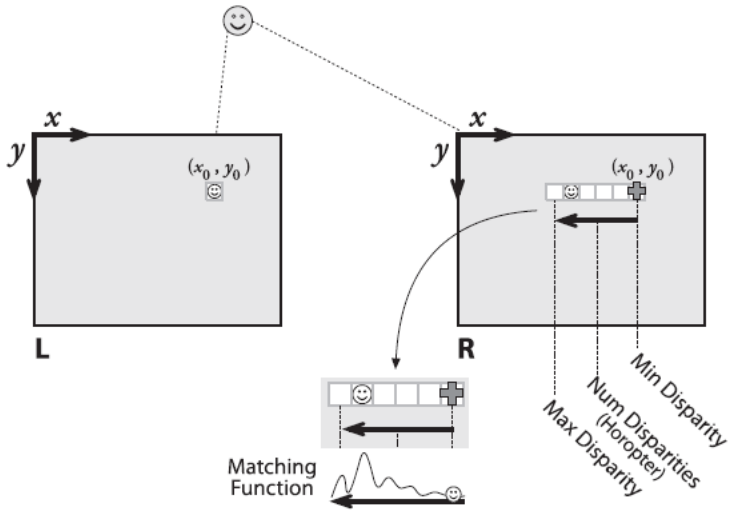


Figure 4-1. Matching search process. Source: Bradski, G. and Kaehler, A. (2008). Learning OpenCV. O'Reilly Media, Inc., Sebastopol, California. ISBN 978-0-596-51613-0.

Another frequently used post-filter is an interest operator which assigns high confidence value to areas which are highly textured. In OpenCV's block matching we employ a parameter called textureThreshold to reject any match that is below the texture threshold, thus avoiding ambiguous matches in flat areas. Finally, to handle discontinuities at object boundaries, a small speckle window is optionally applied to control disparity variation within that window.

Key parameters that need to be tuned when using block matching algorithm are summarized as below:

preFilterSize – Defines the window size for the averaging operator. Its value varies from 5x5 to 21x21. The pre-filter normalizes pixel's intensity with the average intensity within its window. The purpose of this operator is to reduce light disturbances and enhance texture.

SADWindowSize – Defines the size of both the support window in the left image and the correlation window in the right image. Window sizes are odd numbers like 9x9, 21x21, 41x41, etc. A smaller window size might increase the number of mismatches while a bigger window slows down the computation while giving a smoother disparity map.

minDisparity – Minimum disparity decides the starting position for the matching process. Its value defaults to 0, meaning that the search in the right image will start at exactly the same pixel position as in the left image.

numberOfDisparities – This value controls the searching range along scan line. Since disparity is inversely proportional to object distance, usually a smaller value of numberOfDisparities is chosen if we want to see things far away, while a larger value is chosen if we want to see things close up. To estimate the initial value of numberOfDisparities, one can insert baseline T and camera focal length f into equation 4-4

$$Z = \frac{fT}{x_r - x_l} \tag{4-4}$$

Where Z represents distance of target and $x_r - x_l$ represents disparity in pixels.

textureThreshold – Disparity is only computed for pixels within a textured enough region. Its default value is 12. This prevents any ambiguous match in flat areas.

uniquenessRatio – It controls the screening criteria for disparity candidates. A strong candidate should have a global minimum SAD value within the whole search range. The default value of uniquenessRatio is 12.

User Interface

To better understand how these parameters influence the result of disparity map, a user interface which allows the user to adjust the value of parameters in real time is designed in this research. OpenCV has a built-in function called "cvCreateTrackbar". It creates a slider and assigns a value to be a position synchronized with the slider. In this research each of the above mentioned parameters is attached to a slider that can be controlled by the user. Whenever the user updates the slider, its value is applied to the corresponding parameter that accepts the update periodically. Figure 4-2 shows the user interface together with the disparity map computed using the Block Matching algorithm. Meanwhile, Figure 4-3 shows the comparison among disparity maps resulting from different SAD window sizes.

Image Segmentation Based Foreground Extraction

For obstacle avoidance what we are truly concerned with is the distance and size of the obstacle itself. A complete dense disparity map containing all depth information of the image captured by the camera will generate redundant information. This makes it more difficult in post-processing such as distance calculation or 3D reconstruction of the target object. Hence we try to find a method that could extract foreground objects without consuming too much computation power.

Many techniques have been developed for image segmentation. The fastest approach is to

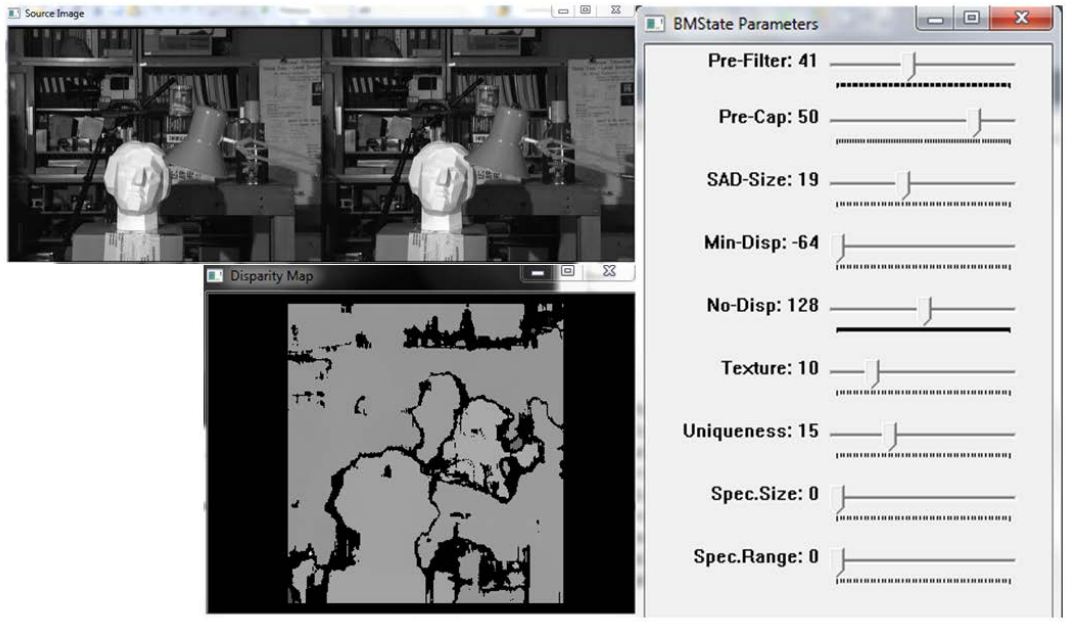


Figure 4-2. User interface for block matching algorithm

Figure 4-3. Disparity maps derived from different SAD window size. Smaller SAD gives more noise yet captures object contours better. Bigger SAD yields smoother image yet loses contour recognition significantly.

find a threshold that divides the image intensities into two classes. Pixels whose intensity is lower than the threshold will be rejected. This technique assumes that foreground object has a distinctive color as compared to its background, which is true for most vehicle applications like highway (Figure 4-4) or the International Ground Vehicle Competition.

Figure 4-4. Obstacle extraction on a highway. Source: CMU Image Data Base. Reprinted with permission from CMU Image Data Base, http://vasc.ri.cmu.edu/idb/ (Dec 21, 2013).

Since the threshold is an image dependent value that cannot be universally defined, an adaptive approach to compute the threshold needs to be introduced. In this research Otsu's method [Otsu 79] is selected. Otsu's method was first introduced by the Japanese scholar Ōtsu Nobuyuki in 1979 and worked best in practice for images containing two classes of pixels or bi-modal histograms.

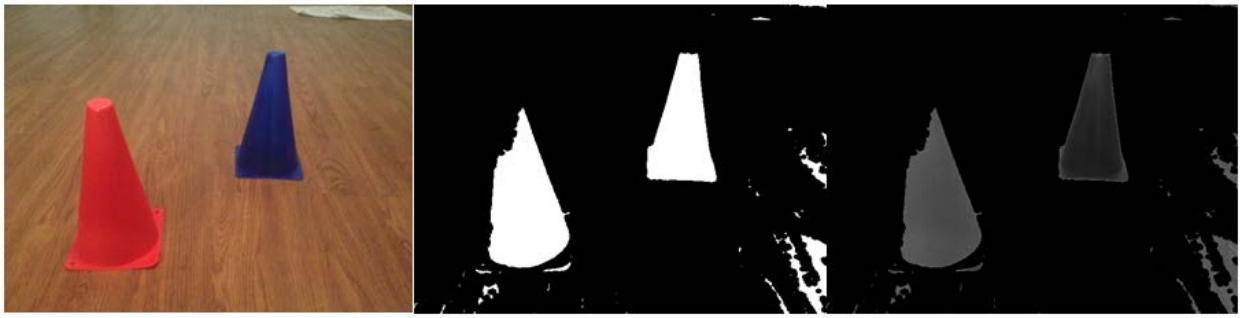


Figure 4-5. Obstacle extraction with single-color background. Photographer: Tianyu Gu. Taken on Jan 27th, 2014 at Gainesville, FL. Only the construction cones are extracted from the background.

Figure 4-6. Obstacle extraction with multi-color background. Photographer: Tianyu Gu. Taken on Jan 27th, 2014 at Gainesville, FL. Both construction cones and the floor are categorized as foreground

To achieve the best result we take four steps to extract an obstacle:

1) Apply Gaussian smoothing to reduce the noise in original image. This is done by convolving each pixel in the image with a Gaussian kernel.
2) Compute threshold using Otsu's method and produce a binary image. This is achieved by exhaustively searching for the threshold that maximizes the inter-class variance.
3) Apply a morphological closing operation to eliminate downward outliers in the image. This is done by first dilating and then eroding the image.
4) Extract obstacles by applying the binary image as a mask to the original image.

Gaussian Smoothing

Since Otsu's method is a statistical approach to segment an image. It is very sensitive to noise which disturbs the distribution of the pixel histogram. Therefore smoothing is necessary as a pre-processing step. Our experiments show that Gaussian Smoothing yields a robust results. A two-dimensional Gaussian function is given by:

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\left[(x-\mu_x)^2+(y-\mu_y)^2\right]/(2\sigma^2)} \quad (4\text{-}5)$$

Where μ_x and μ_y are the mean value in the x and y directions respectively, and σ is the variance that controls the width of the Gaussian distribution.

Unlike the averaging filters which replace each pixel with the average of its neighbors, the Gaussian filter rejects any high frequency artifacts in that it assigns a bigger weights to the nearest neighbors while a smaller weights are assigned to more distant neighbors. Such a smooth reduction in the influence of neighboring pixels avoids any discontinuities in the image.

In reality, images are discrete. Therefore a Gaussian kernel that convolves with each pixel is no longer a continuous function as discussed above. An isotropic discrete Gaussian kernel is shown in Figure 4-7. Note that the kernel has to be normalized so that the pixel to be convolved will preserve the same intensity scale. Finally the convolution is given by:

$$I(x, y) = i(x, y) * G_\sigma \quad (4\text{-}6)$$

Where $i(x, y)$ is the intensity of pixel (x, y) and "*" is the convolution operator.

Image Segmentation Based on Otsu's Method

Otsu's method performs an adaptive thresholding based on the histogram of the grayscale image. It treats an image as foreground class and background class, and computes the threshold value that maximizes the inter-class variance. Table 4-1 shows the parameters involved in Otsu's method. The implementation of Otsu's method is explained as follows.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

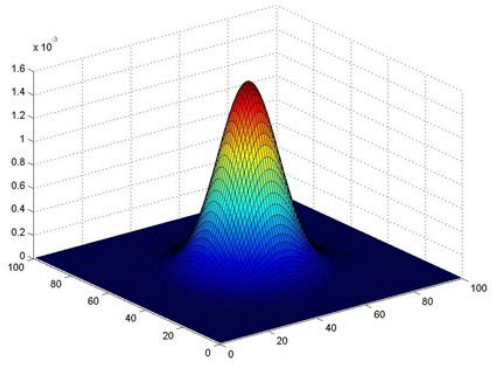


Figure 4-7. Two dimensional discrete Gaussian kernel

Table 4-1. Parameters of Otsu's method

Parameter	Representation
t	threshold value
ω_0, ω_1	Foreground / background class probability
μ_0, μ_1	Foreground / background class mean gray level
μ	Mean gray level of entire image
g	Inter-class variance
$M \times N$	Image size
N_0, N_1	Number of pixels intensity lower / higher than threshold

First, class probability is calculated as:

$$\omega_0 = \frac{N_0(t)}{M \times N} \tag{4-7}$$

$$\omega_1 = \frac{N_1(t)}{M \times N} \tag{4-8}$$

The mean gray level of an image can be expressed as

$$\mu(t) = \omega_0(t) \times \mu_0(t) + \omega_1(t) \times \mu_1(t) \tag{4-9}$$

Then, the definition of inter-class variance is given by

$$g(t) = \omega_0(t)[\mu_0(t) - \mu(t)]^2 + \omega_1(t)[\mu_1(t) - \mu(t)]^2 \tag{4-10}$$

Substituting μ into equation 4-10 yields

$$g(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \tag{4-11}$$

Based on eq.5, Otsu's algorithm will first compute the histogram of the input image, and traverse all possible values of threshold t from 1 to maximum intensity until a global maxima of inter-class variance $g(t)$ is achieved. Once the threshold is determined, the algorithm will convert the input image to a binary image with foreground class pixels being 1 and background class pixels being 0.

Morphological Closing Operation

To further improve the results from thresholding, a morphological closing operation is applied to connect bright regions while at the same time retaining their basic size. Morphological

closing is achieved by first dilating the image and then eroding the image. Dilation is first applied to expand bright regions so that an object fills most of its concavities. Later erosion is applied to smooth away upward outliers and reduce the object size to its basic size.

Similar to Gaussian smoothing, both dilation and erosion are achieved by convolving the original image $I(x, y)$ with a box-sized kernel$K$. For the dilation operation, kernel$K$ will compute a local maximum value of pixels beneath it and replace the center pixel with the local maxima. While for the erosion operation, kernel$K$ will compute a local minimum value instead.

Foreground Extraction

Finally, the extraction of the foreground object according to the threshold binary image is straightforward. We treat the binary image derived from Otsu's method as a mask that filters out pixels below the threshold intensity. Such an operation is depicted as Figure 4-8.

Performance Analysis

This section analyzes the performance of the obstacle extraction algorithm with varying image scenes. Table 4-2 and 4-3 lists the parameters and testing results.

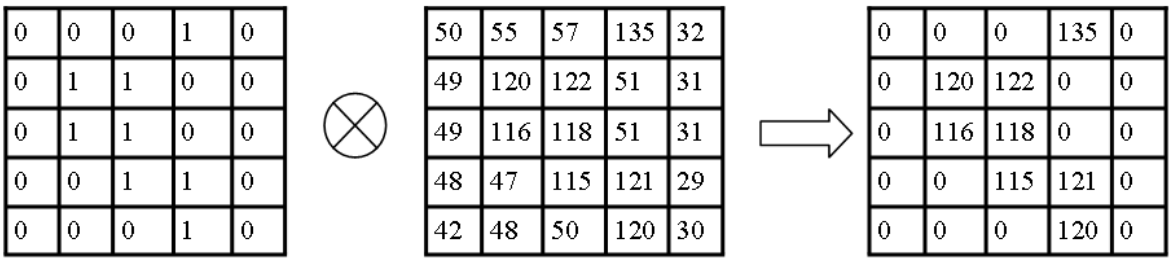


Figure 4-8. Binary mask (left) is applied to original image to produce foreground image

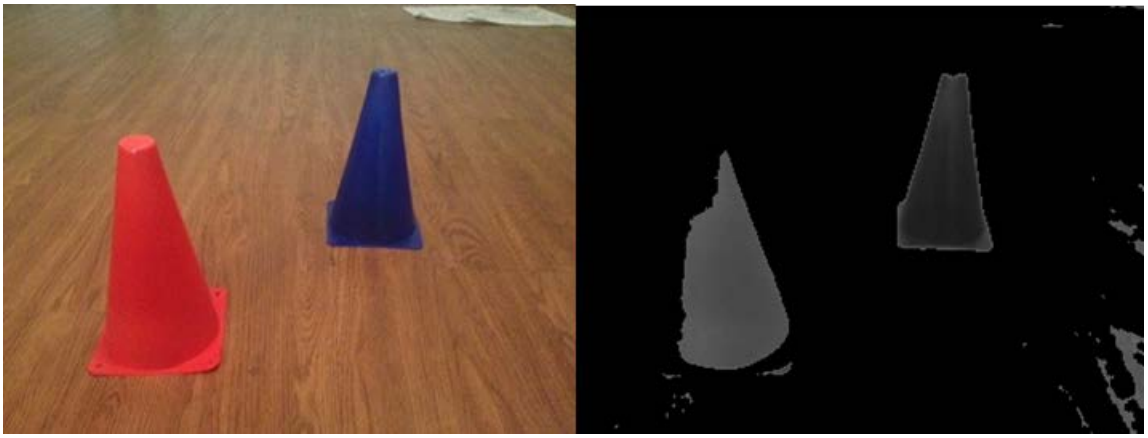
Table 4-2. Parameters for image segmentation test

Parameter	Value
CPU speed	2.5GHz
Gaussian kernel size	3×3
Gaussian σ	1.5
Number of Closing Iterations	2

Test results of image segmentation suggest that our technique is very efficient and robust in terms of computation time. The total processing time of object extraction is roughly 2 ms at a resolution of 320×240 regardless of image complexity. This is ideal for the implementation of real-time stereo correspondence which requires a frame rate of 30 fps. However, this technique is unstable in terms of the foreground object extraction. For images with single-color background, this technique could easily segment the foreground. For images with two or more background colors, it will only recognize one background that has the most distinctive brightness, and the rest of the image will be recognized as foreground object. For images containing foreground objects similar to the background, this technique will also fail to extract them. Figure 4-9 shows examples of the segmentation process.

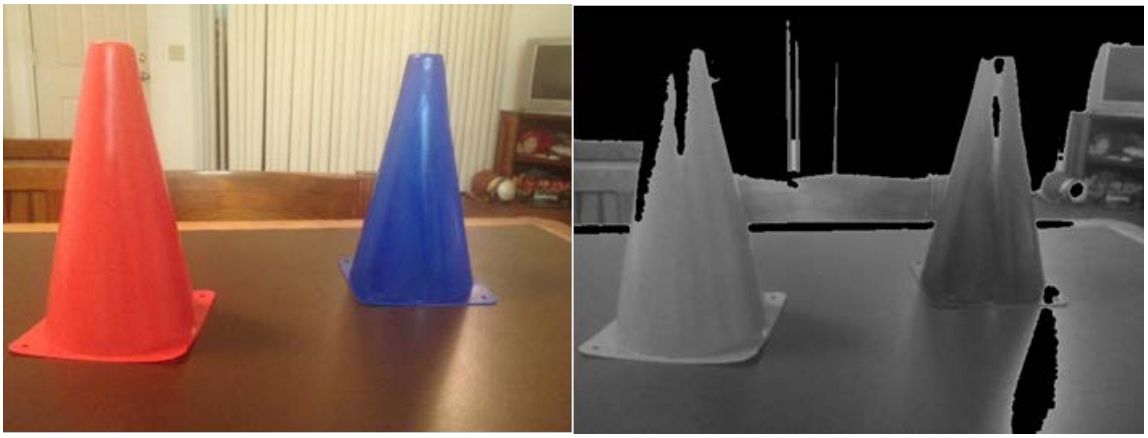
Table 4-3. Image segmentation speed comparison

Test Image	Parameter	Value
A) Single-color Background	Resolution	320×240
	Total Time Spent	1.5 ms
B) Bi-color Background	Resolution	240×354
	Total Time Spent	2 ms
C) Complex Background	Resolution	320×240
	Total Time Spent	2 ms
D) Multiple Objects	Resolution	320×260
	Total Time Spent	2 ms



A


B


C


D

Figure 4-9. Images before and after image segmentation. A) *Objects with single color background are easily extracted.* Photographer: Tianyu Gu. Taken on Jan 27th, 2014 at Gainesville, FL. B) *Objects with both lawn and sky as backgrounds*. Source: International Ground Vehicle Competition. Reprinted with permission from IGVC

Online, http://www.igvc.org/photos.htm (Jan 17, 2014). Result suggests that only sky is recognized as background. C) *Objects with complex background*. Photographer: Tianyu Gu. Taken on Jan 27th, 2014 at Gainesville, FL. Only curtain is recognized as background. D) *Multiple objects with single color background*. Source: Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1), 7–42. Reprinted with permission from Middlebury Computer Vision Pages, http://vision.middlebury.edu/stereo/data/ (Dec 7, 2013). Paper hat is recognized as background due to similar brightness to wall.

Stereo Correspondence with Foreground Extraction

In this section we are going to apply the image segmentation technique prior to the stereo correspondence. Meanwhile the comparison of the disparity map between segmentation-based correspondence and basic stereo correspondence will be examined in terms of their efficiency, quality, and limitations.

Disparity Result Analysis

First we will look at a pair of images with strong contrast between the foreground object and background. Figure 4-10 shows both image pairs before and after applying Otsu' image segmentation. Basically the sculpture, tree and buildings are grouped as foreground objects while the sky is recognized as background. A detail examination shows that the top of the vertical post has been eroded somewhat after segmentation, which is due to lighting variations on the object surface. The white parts of the buildings have also been grouped as background. Therefore object with strong brightness variations within itself could also result in false segmentation.

Figure 4-10. Rectified stereo image pair before and after image segmentation. Source: CMU Image Data Base. Reprinted with permission from CMU Image Data Base, http://vasc.ri.cmu.edu/idb/ (Dec 21, 2013).

Figure 4-11 shows the comparison between disparity maps computed directly from raw image pairs and segmented image pairs. It can be seen that both of the sculptures have the same level of noise. However, the segmentation-based disparity map shows a very clean background, while the original disparity map has many speckles in the background. Since backgrounds with black pixels in left/ right images would not generate any disparity information, this prevents any noise from being generated during the process of block matching.

Next we apply the same approach to the “Head and Light” stereo image pair as shown in Figure 1-2. Figure 4-12 shows the image pair after image segmentation and corresponding disparity map. It can be seen that both the head and light have been grouped as background since they have similar brightness as the whiteboard in the background. The disparity map therefore has missing disparities on the head sculpture and light, which would not produce any useful depth information during post-processing. This is the situation we want to avoid.

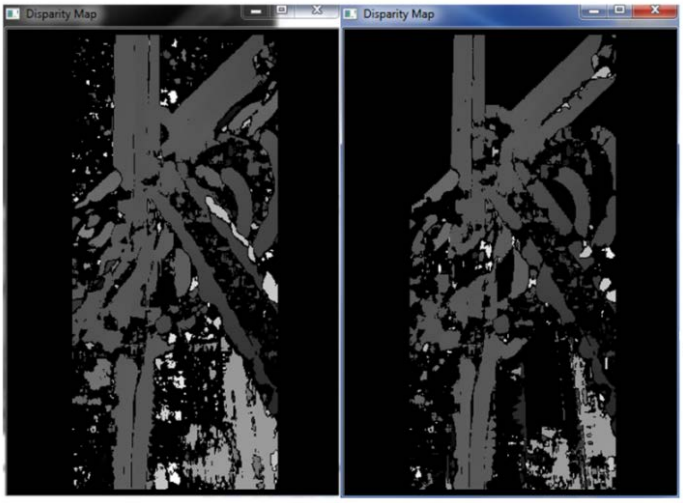


Figure 4-11. Disparity map computed from raw image pairs (left) and segmented image pairs (right). The disparity map on the right has significantly less noise in background areas.

Figure 4-12. Disparity map computed from segmented “Head and Light” image pairs. Source: Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1), 7–42. Reprinted with permission from Middlebury Computer Vision Pages, http://vision.middlebury.edu/stereo/data/ (Dec 7, 2013).

In applications like robot navigation, most of the obstacles have either strong colors or distinctive colors as compared to their background such as sky or road. Therefore in most cases we will get the result similar to Figure 4-10.

Efficiency Analysis

The next thing we are concerned about is whether the combined approach would satisfy real time purposes. For example, if we want to achieve a frame rate of 30 fps, then for each frame the total processing time should not exceed 33.3 ms. Just like the analysis in section 3.1.5, most of the test images will have a total pixel number between 76,000 and 100,000. Figure 4-13 shows the total processing time for each test image. It can be seen that both the original blocking matching algorithm and the segmentation-based approach take less than 25 ms. One can also tell from the figure that the combined approach is even faster than the original approach. Even though image segmentation requires a certain amount of computation power, it reduces the number of pixels to be processed in the block matching algorithm by setting the background pixel value to zero.

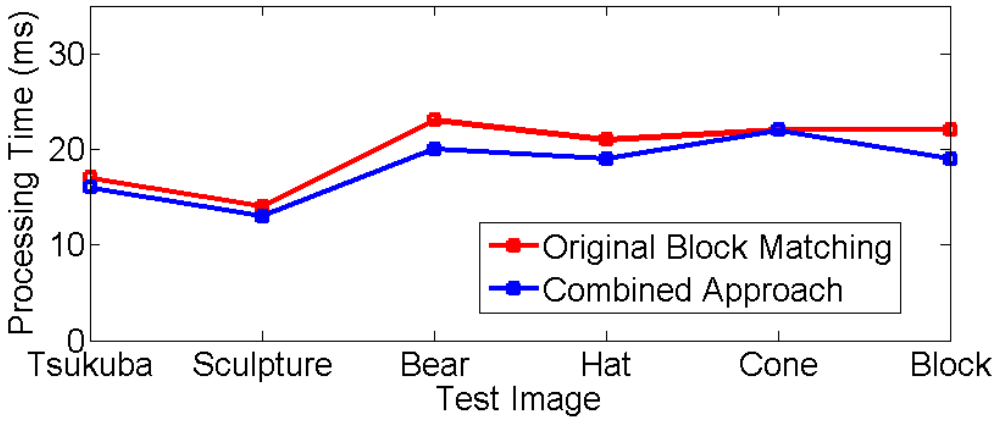


Figure 4-13. Stereo correspondence processing time for each of the test images. The red line represents original block matching method. The blue line represents segmentation based blocking matching method.

CHAPTER 5
IMPLEMENTATION

To apply stereo correspondence to images captured from real cameras, we first need to handle problems like camera synchronization, calibration and rectification. A stereo vision system with bad synchronization would have its left and right cameras capturing different images especially on a moving platform. Calibration and rectification are necessary to make the input images co-planar and row-aligned.

Image Capture

The interface with webcams is achieved via the CL-Eye Platform Software Development Kit (SDK). To maximize camera synchronization, a new thread is created alongside the main function to handle all camera captures and data transfer between the camera and host computer. Inside the capture thread, we first create an instance for each of the cameras based on its unique camera ID. Meanwhile a data buffer is used to retrieve frame data from specified camera instances. Once the buffer captures a frame, it sends the frame to its corresponding image instance (left or right) immediately for stereo processing. Thanks to the built-in mechanism designed by Sony, all image data are transferred without any processing and thus guarantees raw image quality and capture rates. The capture process is looping at a frequency controlled by the user-specified frame rate. Since each of these two operations runs very fast with only O(1) complexity, the continuous capturing of left and right images can be viewed as almost simultaneous. Figure 5-1 illustrates the camera capture process.

Stereo Calibration

Stereo calibration is the most crucial part during the entire process since it produces key parameters for stereo rectification. Similar to monocular camera calibration which finds the function that maps 3D object points into a 2D image plane, stereo calibration aims at finding the

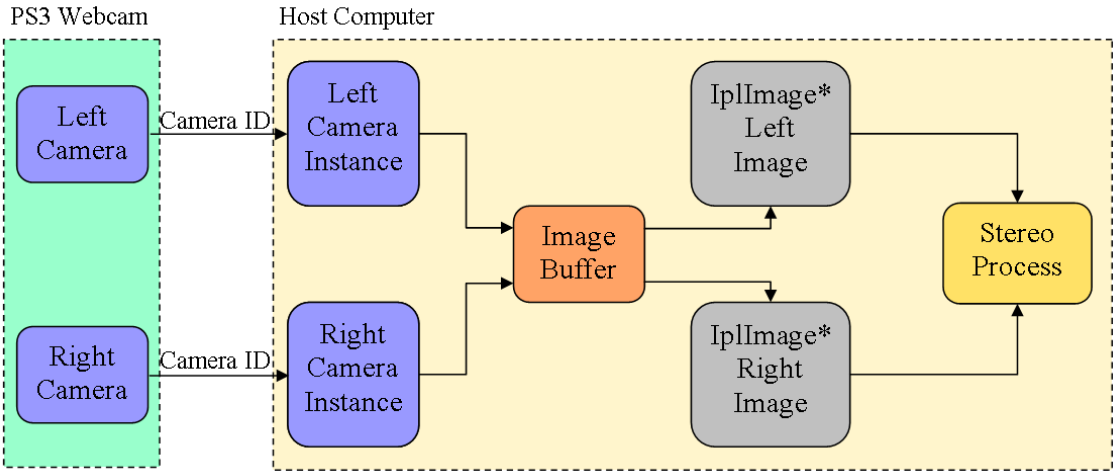


Figure 5-1. An independent thread is initialized to capture images from left and right cameras simultaneously.

geometrical relationship between the left and right cameras, which is given by the rotation matrix R and translation T. They relate two cameras through the following equation:

$$P_r = R(P_l - T) \qquad (5\text{-}1)$$

Where P is the object point that seen by both cameras, and P_r , P_l denotes the point's coordinates with respect to the right and left camera coordinates respectively. Once we have this relationship, we could align left and right cameras so that they are in frontal-parallel configuration depicted as Figure 1-1.

Stereo calibration is implemented using OpenCV's "cvStereoCalibrate" function. First a 6×8 chessboard pattern is printed on a letter sized paper that is firmly attached to a hard-covered book. Then we place the chessboard pattern in front of the cameras so that both cameras can read all 48 corners. Once the calibration starts, we start to move and rotate the chessboard, allowing both cameras to see the chessboard from different viewpoints. Experiments show that a total of ten (10) images of the chessboards would be sufficient for calibration. Before calling the function "cvStereoCalibrate", we need to use "cvFindChessboardCorners" to locate the corners' position in terms of pixel coordinates. In addition, the value returned by

"cvFindChessboardCorners" are only approximate, thus we further need to call "cvFindCornerSubPix" in order to refine our results to sub-pixel accuracy. Finally, "cvStereoCalibrate" will take those information as input and compute for rotation matrix, translation vector, camera intrinsic matrices, essential matrix, and fundamental matrix. Figure 5-2 shows one moment of the stereo calibration process.

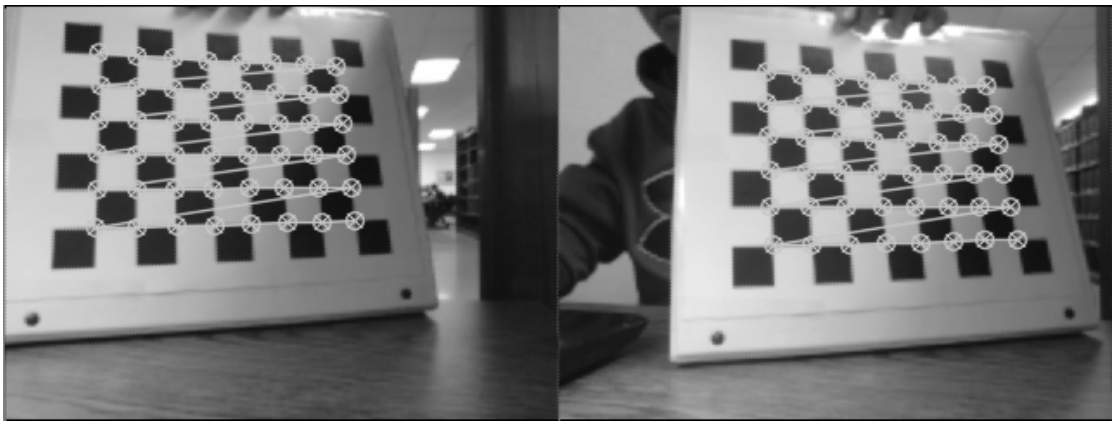


Figure 5-2. Stereo calibration using a 6×8 chessboard pattern. All 48 corners are located and drawn in the camera view. Photographer: Tianyu Gu. Taken on Feb 7th, 2014 at University of Florida, Gainesville, FL.

Stereo Rectification

Even though stereo cameras are aligned manually, in most cases the image planes are not perfectly coplanar or row-aligned. Thus epipolar geometry cannot be applied to stereo correlation. Stereo rectification is a process that rectifies both images so that they are co-planar and row-aligned. During stereo calibration, we have already computed rotation matrix R that will rotate the right camera's image plane into the left image plane. However further operation is needed to make both images row-aligned. The rectification technique we use was first presented by Tsai [Tsai 87] and later simplified by Jean-Yves Bouguet [BouguetAlg] through his implementation in MATLAB's calibration toolbox. Basically Bouguet's method uses the rotation matrix R and translation vector T computed from calibration to rectify both images. According to epipolar geometry introduced in Chapter 1, if horizontal rows are aligned in a same

plane, then the epipoles will be located at infinity, also known as “ideal points”. However, such a constraint is insufficient to perform our calculation since there are infinite parallel image planes. Bouguet’s method actually adds one more constraint by attempting to minimize re-projection distortion. First, the rotation matrix R is split in half between left and right cameras, and two resulting rotation matrix for left and right camera is denoted as r_l and r_r respectively. If we rotate each camera by their respective rotation matrix, then we are actually making both cameras coplanar. Later, row-alignment is achieved by finding another rotation matrix R_{rect} that pushes the left camera’s epipole to infinity and align both epipolar lines horizontally. Denote R_{rect} as

$$R_{rect} = \begin{bmatrix} (e_1)^T \\ (e_2)^T \\ (e_3)^T \end{bmatrix} \tag{5-2}$$

Next, we will solve for e_1, e_2, e_3 respectively. Let’s define the principal point (c_x, c_y) as left image’s origin (Principal point is where principal ray intersects with image plane). In Bouguet’s method, it chooses the first vector as the direction of the epipole e_1. Since e_1 is located at infinity, the first vector can be expressed as the normalized translation vector between two cameras:

$$e_1 = \frac{T}{\|T\|} \tag{5-3}$$

For a rotation matrix, the three vectors must be orthogonal to each other, thus we arbitrarily choose e_2 as a vector both orthogonal to the principal ray and e_1. Taking the cross product of principal ray and e_1 and normalizing the result yields:

$$e_2 = \frac{[-T_y \quad T_x \quad 0]^T}{\sqrt{T_x^2 + T_y^2}} \tag{5-4}$$

And e_3 is computed by taking the cross product of e_1 and e_2:

$$e_3 = e_1 \times e_2 \tag{5-5}$$

Finally, rectification is achieved by first aligning the planes and then aligning the rows:

$$\mathrm{R_l} = R_{rect} r_l \tag{5-6}$$

$$\mathrm{R_r} = R_{rect} r_r \tag{5-7}$$

We choose Bouguet's rectification technique in this research because it produces less distorted images and computes for a re-projection matrix Q that can be used for depth calculation or 3D reconstruction. To start the rectification process, we first call OpenCV's function "cvStereoRectify", which takes as input the rotation matrix R and translation vector T that was computed during the calibration process, and returns the rectification rotations $\mathrm{R_l}, R_r$ and projection equations $\mathrm{P_l}, P_r$. Then, instead of performing the calculation every time the computer grabs a frame, we create lookup maps that project raw images into rectified images. Lookup maps are created by separately calling the function "cvInitUndistortRectifyMap" for left and right images immediately after calling "cvStereoRectify". Finally, rectification is performed every time we captured a new pair of images by calling "cvRemap", which takes the lookup maps as input and produces rectified images that are horizontally aligned with each other. Figure 5-3 shows the comparison between raw input images and rectified images.

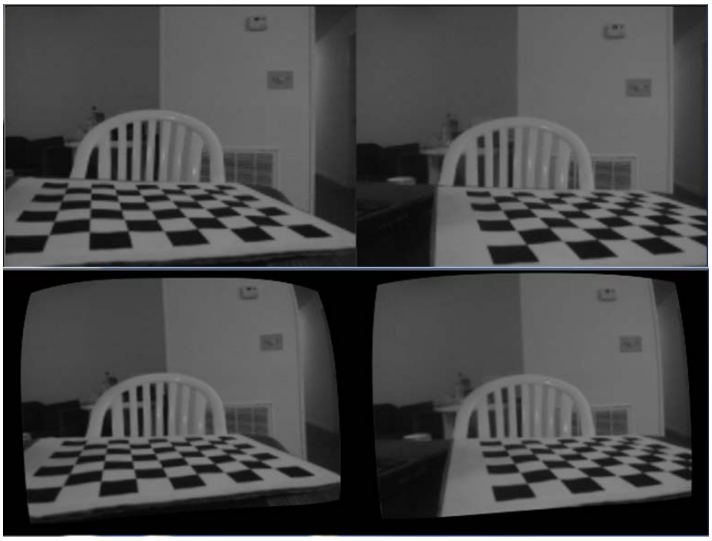


Figure 5-3. Raw input images (top) and rectified, row-aligned images (bottom). Photographer: Tianyu Gu. Taken on Feb 9th, 2014 at Gainesville, FL.

3D Reconstruction and Distance Calculation

Once we have derived the disparity d for each image pixel (x, y) and camera matrices, the computation that re-projects the 2D image pixel to 3D coordinates is very straightforward. If we compute the 3D coordinates of an object's centroid, then we can estimate how far away the object is. If we could compute 3D coordinates of all pixels in the disparity map, then we are actually reconstructing the environment. During the stereo reconstruction process, the function "cvStereoRectify" optionally returns a 4 by 4 re-projection matrix Q that contains all necessary parameters for 3D coordinates computation:

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X_h \\ Y_h \\ Z_h \\ W \end{bmatrix} \tag{5-8}$$

where X_h, Y_h, Z_h are the homogeneous 3D coordinates of the object point. The actual 3D coordinates are then given by:

$$X = X_h / W \tag{5-9}$$

$$Y = Y_h / W \tag{5-10}$$

$$Z = Z_h / W \tag{5-11}$$

CHAPTER 6
EXPERIMENTAL RESULTS

Stereo calibration and rectification produces co-planar, row-aligned image pairs that could be processed by block matching algorithm introduced in Chapter 4. In this chapter we will examine the results of block matching stereo correspondence and segmentation-stereo combined approach that are applied to real time cameras.

Stereo Correspondence Parameter

The block matching algorithm has many parameters, as introduced in chapter 4. When we are designing a stereo vision system for a mobile robot, it would be ideal if we could find a set of parameter configuration that works nicely for most situations. By testing different values of block matching parameters, we try to determine a set of parameters that maximize the robustness of our system.

SAD Window Size

The first parameter we are going to examine is the window size of SAD (Sum of Absolute Differences). Previous experiments suggest that a smaller SAD window brings more noise while a bigger SAD slows down the computation. To satisfy real-time computation speed without sacrificing too much quality, we seek to find one or several SAD window sizes that strike a balance between computation speed and noise level.

Figure 6-1 shows the rectified stereo image pair together with disparity map generated with different SAD window size. It can be seen from the figure that while a smaller window size brings more noise, it gives a more accurate contour of the cartoon model. A SAD window size of 13 (leftmost) produces much noise that we want to avoid in the first place. By experimenting with different small SAD window sizes around 13, we conclude that a minimum SAD window size of 17 produces the disparity map with an acceptable noise level.


A


B

Figure 6-1. Comparison of disparity map using different SAD window sizes. A) *Rectified stereo image pair*. Photographer: Tianyu Gu. Taken on Feb 16th, 2014 at Gainesville, FL. B) *Disparity map generated with different SAD window size*. SAD size from left to right: 13, 21, 37.

To determine the upper limit of SAD window size, we start to look at the time it takes to perform a complete stereo correspondence task for the image pair. Recall that our goal is to limit our computation time to 33ms in order to achieve a 30Hz frame rate. Therefore any window size exceeding the required time will be rejected. For the same scene as Figure 6-1, we experiment with different SAD window sizes and see where it fails to meet the time requirement. Figure 6-2 illustrates the time it takes to process stereo correspondence with different SAD window sizes under fixed disparity search range.

According to the figure, the processing time grows very slowly when SAD window size is smaller than 21. However the time substantially increases when we reach a window size of 23, which far exceeds the time limit shown as the red line.

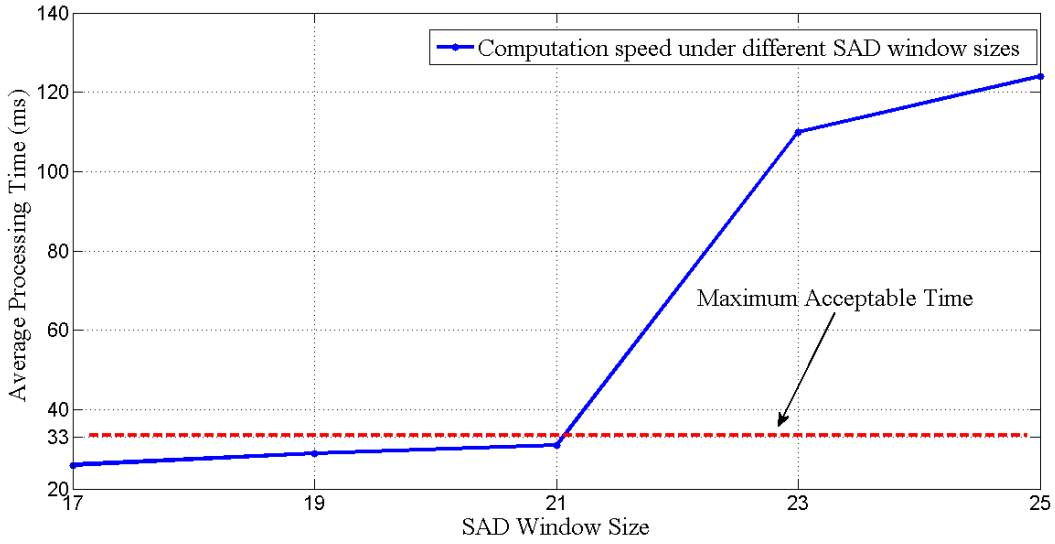


Figure 6-2. Average stereo processing time under different SAD window sizes.

If we combine our analysis from the above two perspectives, the acceptable SAD window size would be 17, 19, or 21. We could further narrow down the range by testing these values over other image sets. Table 6-1 lists other parameters that are fixed for our experiment.

Table 6-1. Parameters for SAD window size test

Parameter	Value
Minimum Disparity	-64
Number of Disparities	128
Pre-filter Cap	31
Uniqueness Ratio	15
CPU Speed	2.5 GHz

Disparity Search Range

Since computation speed is not only sensitive to SAD window size but also to disparity search range, the next parameters we are going to examine are Minimum Disparity and Number of Disparities. A wider search range gives a better resolution for things that are close up, yet it at the same time slows down the computation. Therefore we will test these two parameters from the following aspects: Computation speed and minimum depth resolution.

First we will fix the SAD window size to 21 and run our program with different search ranges. Figure 6-3 shows the speed versus different search ranges.

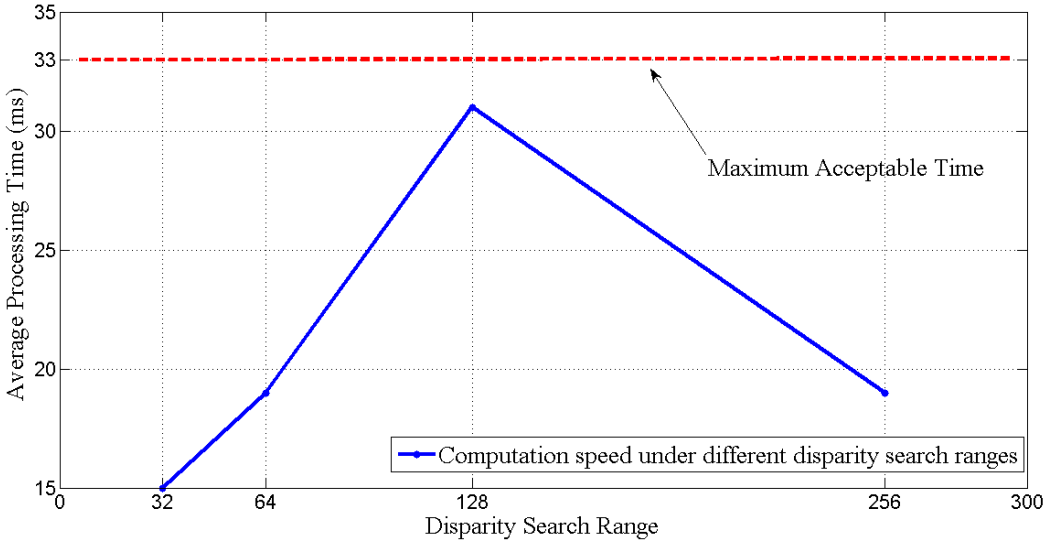


Figure 6-3. Average stereo processing time under different disparity search ranges.

From the figure we can see that a small search range (32 pixels) consumes only half of the time of a relatively large search range (128 pixels). It is interesting to see that when we further increase the search range to 256 pixels, the time surprisingly drops. This is because our disparity map becomes much narrower when we substantially increase the disparity search range, thus reducing the total processing workload. Figure 6-4 shows the cropped disparity map at a disparity search range of 128.

Next we will examine the minimum depth resolution (how close the system can see things) under different disparity search ranges. The same cartoon model will be moved toward the camera along the optical axis to see where it disappears. A search range of 256 can be discarded since it produces a narrow disparity map. Thus we will run our program only with a search range of 32, 64 and 128. Table 6-2 shows their comparison in terms of depth resolution. Similarly, all the other parameters for block matching algorithm are fixed and listed in Table 6-3.

Figure 6-4. Disparity map is heavily cropped when a large disparity search range is used. Photographer: Tianyu Gu. Taken on Feb 16th, 2014 at Gainesville, FL.

Table 6-2. Depth resolution under different disparity search range

Disparity Search Range	Minimum Depth
32	540 mm
64	360 mm
128	200 mm

Table 6-3. Parameters for disparity search range test

Parameter	Value
SAD window size	21
Pre-filter Cap	31
Uniqueness Ratio	15
CPU Speed	2.5 GHz

Experiments show that a disparity search range of 32 pixels is not capable of seeing things that are closer than 540mm. While a search range of 128 can see things as close as 200mm, which dramatically increases the system's range to resolve depth. Therefore, disparity search ranges of 64 and 128 will be recommended in our research since they both demonstrate fast computation speed and high depth resolution.

Other Parameters

While "Pre-filter Cap" and "Uniqueness Ratio" do not impact the computation speed of stereo correspondence, our experiment will further show that a proper value for each of these

parameters will enhance the quality of disparity map. A pair of test images was taken in an indoor environment with controlled lighting. The rectified image pair is shown in Figure 6-5.

Figure 6-5. Rectified image pair for testing pre-filter cap and uniqueness ratio. Photographer: Tianyu Gu. Taken on Feb 16th, 2014 at Gainesville, FL.

Pre-filter Cap – We experimented with different values for this parameter and picked three of them to show in Figure 6-6. The results show that a bigger pre-filter cap produces a brighter disparity image with less noise. Generally speaking, a pre-filter cap between 50 and 63 (the maximum) would usually produce a cleaner image than those with smaller values.

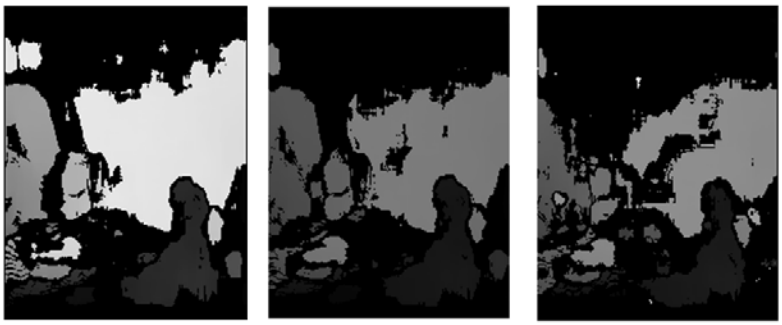


Figure 6-6. Disparity map using different pre-filter cap sizes. A bigger cap brings less noise and increases the brightness of the entire image.

Uniqueness Ratio – The value of this parameter determines how strong the winning pixel should be as compared to its neighbors. Therefore the uniqueness ratio should be large enough to filter out any suspicious matching, and yet small enough so that it does not throw away any good match. Figure 6-7 shows the comparison among different uniqueness ratio. The disparity map with a ratio of 5 contains more noise and detail than others. While the disparity

map with a ratio of 30 contains less noise, it creates more black regions within an object, which is caused by its stringent screening criteria. The experiment also shows that a uniqueness ratio between 10 and 15 usually produces a smooth disparity map without wiping out too much detail.

Uniqueness Ratio = 5 Uniqueness Ratio = 10 Uniqueness Ratio = 30

Figure 6-7. Disparity map using different uniqueness ratios.

Finally, Table 6-4 summarizes the optimized configuration for the stereo correspondence parameters.

Table 6-4. Optimized parameters for stereo correspondence

	Parameter	Value
System Requirement	CPU speed	2.5 GHz
	Image Resolution	320 x 240
	Frame Rate	30 fps
Recommended Value	SAD Window Size	17, 19, 21
	Disparity Search Range	64, 128
	Pre-filter Cap	50 to 63
	Uniqueness Ratio	10 to 15

Outdoor Depth Perception

The scene we use for outdoor experiments is an environment with several construction cones placed at different locations. In figure 6-8 A), both cones are very close in the direction of Z axis (optical axis), therefore their brightness in the disparity map looks the same. As we move the right cone towards to the camera while we move the left cone away from the camera, the right cone gets darker and left cone gets lighter as shown in B).

Another phenomenon one could observe is the difference in the lawn area. In figure A) stereo matching hardly finds any correlation between left and right lawn area, thus producing a black region in the disparity map. However when the lawn gets brighter due to varying lighting conditions, stereo matching starts to find some sparse correlation that results in the white and grey speckles in figure B). To avoid the unstable performance caused by changing ambient light one need to use auto-iris cameras [Ahmed 06] that is capable of adjusting the aperture size according to the lighting conditions. For now, we just need to know the cause of this problem.

Figure 6-9 further shows the result of other situations. In figure A) the disparity map can easily tell two different objects from their brightness even though they overlap with each other to some extent. Figure B) shows depth perception of multiple objects at different locations.

Figure 6-8. Disparity map in outdoor environment with a SAD window size of 21. A) *Both of the two cones are close to the camera.* Photographer: Tianyu Gu. Taken on Feb 9th, 2014 at Gainesville, FL. B) *Right cone is closer to the camera while left cone is more distant.* Photographer: Tianyu Gu. Taken on Feb 9th, 2014 at Gainesville, FL. The darker the pixel in the disparity map, the closer the object is. Black pixels are those rejected during stereo correspondence.

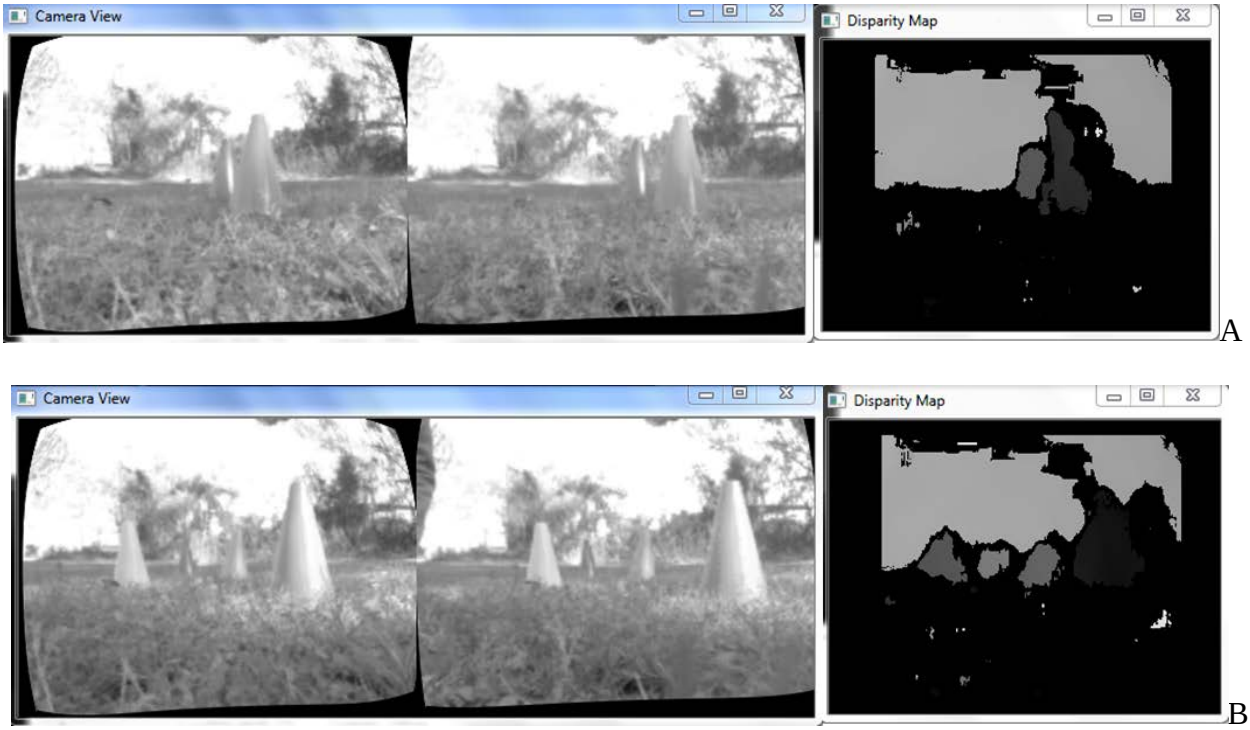


Figure 6-9. Disparity map in outdoor environment. Photographer: Tianyu Gu. Taken on Feb 9th, 2014 at Gainesville, FL.

Comparison between Original Method and Combined Method

Our experiments show that the segmentation-stereo combined approach works poorly in the outdoor environment due to varying ambient light that cannot be handled very well by generic webcams. In our research the comparison between original block matching and combined approach will be limited to the indoor environment.

The scene (figure 6-10) we choose contains a blue construction cone as our main target. According to our analysis in Chapter 4, any image with two or more backgrounds will have the "weakest" one being filtered out. Therefore one could expect that in this scene the door and curtain will be treated as backgrounds while the construction cone and the floor will be extracted. Figure 6-10 shows the snapshots from both cameras and the disparity maps derived from both

methods introduced in Chapter 4. The result shows that the combined approach produces a target-specific depth map while the original method contains background depth information that makes our target less recognizable. In addition, it can be observed that the combined approach makes the contour of the construction cone sharper. Recall that the local dense stereo correspondence algorithm suffers at object boundaries since it is making smoothness assumptions everywhere. However, if the brightness of background pixels around the object becomes zero, disparity discontinuities will no longer impact the local matching cost. This is the reason why certain cone edges look smooth when the background color is black.

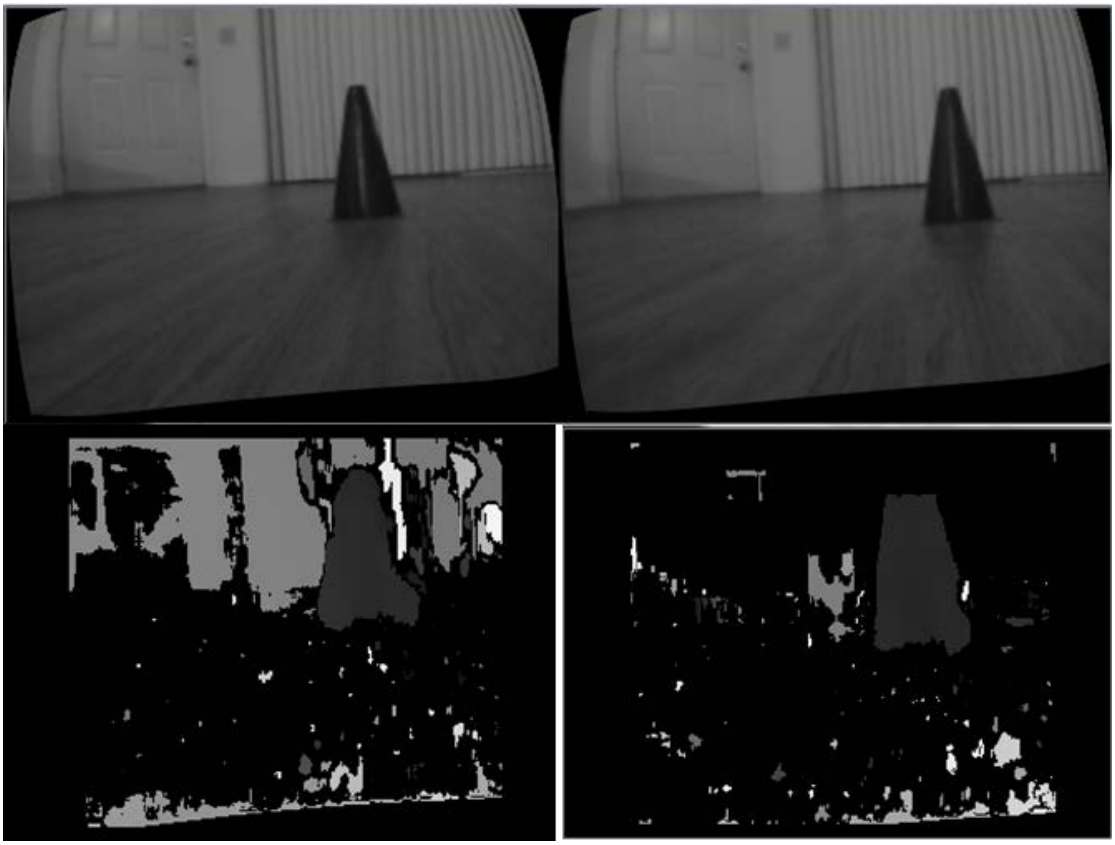


Figure 6-10. Disparity maps computed from original block matching algorithm (left) and segmentation-stereo combined method (right). Photographer: Tianyu Gu. Taken on Feb 2th, 2014 at Gainesville, FL. Both of them use a SAD window size of 17. The original disparity map contains depth information of construction cone and door, curtain behind it. The combined approach successfully extracted construction cone from background, making it easier for object-specific depth calculation.

CHAPTER 7
CONCLUSION AND FUTURE WORK

Conclusions

The focus of this study is comprised of two parts: a) To develop a low-cost real-time stereo vision system capable of perceiving depth at 30 fps; b) To simplify disparity map by applying foreground extraction prior to stereo correspondence.

First, some state-of-the-art stereo correspondence algorithms were reviewed and the local dense stereo correspondence method was preferred due to its fast implementation. Then a pair of PS3 webcams coupled with a laptop were selected as our hardware. Meanwhile computation was performed to find the suitable image resolution for real time purposes. Next, the block matching algorithm was introduced and a user interface was designed to tune key parameters for this algorithm. To achieve our second goal, we further introduced an image segmentation technique based on Otsu's method and analyzed its performance both as a standalone method and a pre-processing step for stereo correspondence. Results showed that this technique works fine as long as there is a visible difference between foreground and background colors.

Finally we applied our stereo correspondence algorithm to real cameras. First, camera synchronization was achieved by creating an independent thread to capture both images simultaneously. Then both stereo calibration and rectification were implemented to make input images co-planar and row-aligned. During the experiments on real-time cameras, different parameters for stereo correspondence were examined and their configuration was optimized to give a better result. Later the system was tested in an outdoor environment to perceive depth of multiple moving objects. Since the lighting conditions are not controlled, our foreground extraction technique failed in the outdoors on the low-cost platform. However, further indoor

experiments were conducted to demonstrate the simplified disparity map with the foreground extraction technique. For all the experiments that we have conducted, computation speed for each correspondence cycle was controlled within 33ms to meet the real time requirement.

Suggestions for Future Work

One of the biggest limitations of binocular stereo vision is the difficulty to achieve both a wide field of view and a deep depth resolution. If we were to improve depth resolution by increasing the baseline distance, then the overlapping area between left and right cameras would have to be sacrificed. While cameras with wider field of view can accommodate both width and depth better, a more appealing solution would be adding one more cameras to form a trinocular stereo vision. A trinocular vision system with real-time performances will further improve the efficiency of depth sensing.

Improvements can also be made from the following two aspects. On the one hand, an auto-iris camera with a better image sensor would enable us to test foreground extraction technique in the outdoors. Machine vision cameras dedicated to outdoor environments will allow more research to be conducted for segmentation-stereo combined approach. On the other hand, more robust algorithms could be developed for foreground extraction techniques so that it is no longer sensitive to the color difference between foreground and background. Such development should not, however, try to improve quality at the cost of increasing computation time.

APPENDIX: STEREO CORRESPONDENCE SAMPLE CODE

```
#include "opencv2/opencv.hpp"
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
#include <limits>
#include "time.h"

using namespace std;

class MyStereo
{
public:
CvMat *imageDepth,*imageDepthNormalized, *dst[2];
IplImage *imageSrc[2], *pDisplayImage;
CvStereoBMState *BMState;
int w,h;

MyStereo()
{
 imageDepth = imageDepthNormalized = 0;
}

void applyClosing( CvMat *binaryImage, int element_radius = 2 )
{
 cvDilate(binaryImage, binaryImage, NULL, 2);
 cvErode(binaryImage, binaryImage, NULL, 2);
}

void createGUI()
{
 BMState = cvCreateStereoBMState();

 BMState->preFilterSize=41; // was 41
 BMState->preFilterCap=50; // was 31
 BMState->SADWindowSize=13; // was 41
 BMState->minDisparity=-64;// was -64
 BMState->numberOfDisparities=128; //was 128
 BMState->textureThreshold=10; //was 10
 BMState->uniquenessRatio=15; //was 15
 BMState->speckleWindowSize=0;
 BMState->speckleRange=0;

 cvNamedWindow( "BMState Parameters", CV_GUI_EXPANDED );
 cvCreateTrackbar("Pre-Filter", "BMState Parameters", &BMState->preFilterSize, 100);
 cvCreateTrackbar("Pre-Cap", "BMState Parameters", &BMState->preFilterCap, 63);
```

```
  cvCreateTrackbar("SAD-Size", "BMState Parameters", &BMState->SADWindowSize, 41);
  cvCreateTrackbar("Min-Disp", "BMState Parameters", &BMState->minDisparity, 50);
  cvCreateTrackbar("No-Disp", "BMState Parameters", &BMState->numberOfDisparities, 256);
  cvCreateTrackbar("Texture", "BMState Parameters", &BMState->textureThreshold, 50);
  cvCreateTrackbar("Uniqueness", "BMState Parameters", &BMState->uniquenessRatio, 50);
  cvCreateTrackbar("Spec.Size", "BMState Parameters", &BMState->speckleWindowSize, 50);
  cvCreateTrackbar("Spec.Range", "BMState Parameters", &BMState->speckleRange, 50);

}

void loadImage()
{
  cvNamedWindow("Source Image", CV_WINDOW_AUTOSIZE);
  cvNamedWindow("Disparity Map", CV_WINDOW_AUTOSIZE);

  imageSrc[0] = cvLoadImage( "left.png", CV_LOAD_IMAGE_GRAYSCALE);
  imageSrc[1] = cvLoadImage( "right.png", CV_LOAD_IMAGE_GRAYSCALE);

  w = imageSrc[0]->width;
  h = imageSrc[0]->height;

//*************************************************************************
***
//*************************************************************************
***

  CvMat *left_temp, *right_temp;
  left_temp = cvCreateMat( h, w, CV_8U ); // added
  right_temp = cvCreateMat( h, w, CV_8U ); // added

  cvSmooth(imageSrc[0], imageSrc[0], CV_GAUSSIAN, 3, 3, 1.5); // Remove noise in disparity
map
  cvSmooth(imageSrc[1], imageSrc[1], CV_GAUSSIAN, 3, 3, 1.5); // Remove noise in disparity
map

  cvThreshold( imageSrc[0], left_temp, 0, 255, cv::THRESH_OTSU |
cv::THRESH_BINARY_INV ); // Using OTSU to threshold the image
  cvThreshold( imageSrc[1], right_temp, 0, 255, cv::THRESH_OTSU |
cv::THRESH_BINARY_INV ); // Using OTSU to threshold the image

  applyClosing(left_temp, 2);
  applyClosing(right_temp, 2);

  dst[0] = cvCreateMat( h, w, CV_8U ); // added
  dst[1] = cvCreateMat( h, w, CV_8U ); // added
  cvCopy(imageSrc[0], dst[0], left_temp);
```

```
  cvCopy(imageSrc[1], dst[1], right_temp);

/*
  cvNamedWindow("Left Image", CV_WINDOW_AUTOSIZE);
  cvNamedWindow("Right Image", CV_WINDOW_AUTOSIZE);
  cvShowImage("Left Image", left);
  cvShowImage("Right Image", right);
  cvSaveImage("left.png", left);
  cvSaveImage("right.png", right);
*/

//*************************************************************************************
***
//*************************************************************************************
***/

  pDisplayImage = cvCreateImage(cvSize(w*2, h), IPL_DEPTH_8U, 1);
  for(int i = 0; i < 2; i++)
  {
    cvSetImageROI(pDisplayImage, cvRect(i*w, 0, w, h));
    cvCopy(dst[i], pDisplayImage);
  }
  cvResetImageROI(pDisplayImage);
  cvShowImage("Source Image", pDisplayImage);
}

void stereoProcess()
{
  clock_t start, finish;
  double duration;
  printf("StereoProcess Start\n");
  start = clock();

  if(!imageDepth) imageDepth = cvCreateMat( h, w, CV_16S );
  if(!imageDepthNormalized) imageDepthNormalized = cvCreateMat( h, w, CV_8U );

  cvFindStereoCorrespondenceBM(dst[0], dst[1], imageDepth, BMState);
  cvNormalize(imageDepth, imageDepthNormalized, 0, 256, CV_MINMAX );

  printf("StereoProcess Done\n");

  cvShowImage("Disparity Map", imageDepthNormalized);
  cvSaveImage("Disparity.png", imageDepthNormalized);

  finish = clock();
  duration = (double)(finish - start) / CLOCKS_PER_SEC;
```

```
  printf( "StereoProcess takes %f seconds\n", duration );
  }
};

int main()
{
  MyStereo *stereo = NULL;
  stereo = new MyStereo();
  stereo->loadImage();
  stereo->createGUI();

while(1)
{
  stereo->stereoProcess();
  cvWaitKey(30);
}

  cvReleaseStereoBMState(&stereo->BMState);
  cvReleaseImage( &stereo->imageSrc[0]);
  cvReleaseImage( &stereo->imageSrc[1] );
  cvDestroyWindow("Disparity Map");
  cvDestroyWindow("Source Image");
  return 0;
}
```

LIST OF REFERENCES

Ahmed, M. F. (2006). "Development of a Stereo Vision system for Outdoor Mobile Robots," M.S. thesis, University of Florida.

Blake, A. and Zisserman, A. (1987). Visual Reconstruction. MIT Press, Cambridge, Massachusetts.

Bouguet, J.-Y. "The calibration toolbox for Matlab, example 5: Stereo rectification algorithm", http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example5.html.

Bradski, G. and Kaehler, A. (2008). Learning OpenCV. O'Reilly Media, Inc., Sebastopol, California. ISBN 978-0-596-51613-0.

Egnal, G. and Wildes, R. P. (2002). Detecting binocular halfocclusions: empirical comparisons of five approaches. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(8):1127 – 1133.

Fusiello, A., Roberto, V., and Trucco, E. (1997). Efficient stereo with multiple windowing. In Proceedings of the 10th IEEE Conference on Computer Vision and Pattern Recognition, pages 858-863.

Gamble, E. and Poggio, T. (1987). Visual integration and detection of discontinuities: the key role of intensity edges. A. I. Memo 970, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6(6), 721–741.

Hannah, M. J. (1974). Computer Matching of Areas in Stereo Images. Ph.D. thesis, Stanford University.

Hartley, R. and Zisserman, A. (2003). Multiple View Geometry in computer vision. Cambridge University Press. ISBN 0-521-54051-8.

Kanade, T. (1994). Development of a video-rate stereo machine. In Image Understanding Workshop, pages 549–557, Morgan Kaufmann Publishers, Monterey.

Konolige, K. (1997). "Small vision system: Hardware and implementation," Proceedings of the International Symposium on Robotics Research (pp. 111–116), Hayama, Japan.

Marr, D. C. and Poggio, T. (1979). A computational theory of human stereo vision. Proceedings of the Royal Society of London, B 204, 301–328.

Otsu, N. (1979). "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62–66.

Roy, S. and Cox, I. J. (1998). A maximum-flow formulation of the N-camera stereo correspondence problem. In Sixth International Conference on Computer Vision (ICCV'98), pages 492–499, Bombay.

Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1), 7–42.

Se, S. and Jasiobedzki, P. (2006). Photo-realistic 3D model reconstruction. In Proceedings of IEEE International Conference on Robotics and Automation, Orlando, Florida.

Sun, J., Zheng, N., and Shum, H. (2003). Stereo matching using belief propagation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(7), 787–800.

Szeliski, R. (2011). Computer Vision: Algorithms and Applications. Springer London Dordrecht Heidelberg New York.

Tsai, R. Y. (1987). "A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," IEEE Journal of Robotics and Automation 3: 323–344.

Zhang, Z. and Shan, Y. (2000). A progressive scheme for stereo matching. In Pollefeys, M. et al., editors, Second European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE 2000), pages 68–85, Dublin, Ireland.

BIOGRAPHICAL SKETCH

Tianyu Gu was born in Chuzhou, Anhui, China. He moved to Hefei, Anhui at the age of 12 and graduated from Hefei No.8 high school in 2006. After receiving his bachelor's degree in mechanical engineering from Shanghai Donghua University in 2010, Gu joined Siemens Ltd., China at Shanghai, China and his major role was mechanical design and product localization of equipment for the iron and steel industry. In the summer of 2012, he left Siemens and continued his education at University of Florida in Gainesville, Florida, USA. He received his master's degree in mechanical engineering with a study focus on robotics and control in May 2014.