A Parameterized Stereo Vision Core for FPGAs

Stephen Longfield, Jr. and Mark L. Chang
Franklin W. Olin College of Engineering
Needham, MA 02492
Email: *mark.chang@olin.edu*

***Abstract*—We present a parameterized stereo vision core suitable for a wide range of FPGA targets and stereo vision applications. By enabling easy tuning of algorithm parameters, our system allows for rapid exploration of the design space and simpler implementation of high-performance stereo vision systems. This implementation utilizes the Census Transform algorithm [1], [2] to calculate depth information from a pair of images delivered from a simulated stereo camera pair. This work advances [3] through implementation improvements, a stereo camera pair simulation framework, and a scalable stereo vision core.**

I. Introduction

In the pursuit of machine vision systems that can perform near the fidelity of biological vision systems, one area of intense study is the acquisition of an accurate three-dimensional model of an unstructured environment. While many machine vision algorithms and systems have sufficed with a two-dimensional understanding of the world around them, advanced robotics, navigation, and the machine intelligence that drives them, would greatly benefit from enhanced spatial perception.

Native three-dimensional perception systems such as LIDAR remain cost- and form-factor-prohibitive for many applications. Fortunately, CMOS camera technology is both inexpensive and favorably packaged for embedded, mobile vision systems. Just as the human vision system creates three-dimensional vision through a pair of eyes that each perceive only two dimensions, stereo vision attempts to infer depth from the disparities between two camera images.

In this paper, we present a significant improvement in the design-time use of stereo vision on FPGAs through a fully parameterized stereo vision core. As a natural extension of our previous work [3], this system enables fast exploration of the design space allowing for better tuning of the stereo vision core to application needs and target hardware resources. This section introduces stereo vision algorithms and related work in hardware acceleration of stereo vision. Section II discusses the hardware and software implementation of our scalable stereo core and presents simulation and synthesis results. Section III details our ongoing and future work.

A. Previous Work

This work extends our previous work [3] through algorithm implementation improvements, a generalized stereo camera pair simulation framework, and by exposing all tunable parameters of the stereo core to the end user for easy modification. As this implementation serves as only the core of larger stereo vision platform, features such as real hardware interfacing with cameras, USB output to a host computer, and host software for post-processing stereo images is not included. However, as each platform's needs are unique, we do not see this as a severe limitation and only note it as a difference from our previous work.

B. Stereo Vision

Reconstructing depth information from a pair of two-dimensional images is an area of active research both in basic algorithms and in their acceleration with hardware. The basic stereo sensing mechanism mimics biological systems with a pair of cameras, separated by some distance, viewing approximately the same "scene". Because of the separation of the cameras, an object will appear to have a lateral shift inversely proportional to its distance from the cameras. You can experience this phenomenon by holding an object close to your face and viewing it with only the left eye, then with only the right eye. The object will appear to have a larger lateral shift the closer it is to your eyes.

By finding the lateral displacement (or disparity, as it is more commonly referred) of an object between camera images, it is simple geometry to calculate the range of an object from the camera. Objects that are relatively far from the camera will occupy nearly the same location in both images, while objects that are relatively close will have a large disparity.

Finding an object in both camera images is called *correspondence*. As described in [4], most correspondence algorithms fall into two basic approaches: area-based or feature-based. Area-based algorithms take windows of pixel intensities in one camera image and attempt to find correspondence in a window of the same size in the other camera image. Feature-based algorithms abstract pixel intensities into features such as edges, corners, lines, contours, or patches, and attempt to find the corresponding features in the other camera image. The potential benefits of feature-based algorithms is that these features will have less variation than raw pixel intensities and will therefore be more robustly correlated across image pairs.

For any of these approaches to be effective, both cameras must deliver images in the same exact global coordinate system. This entails removing or compensating for image distortion, such as rotation, scaling, and vertical translation. This is often accomplished through careful mechanical alignment of the camera pair and a calibration phase to compensate for lens and imager differences.

C. Census Transform

A thorough investigation of a variety of algorithms from [1], [5]–[7] led to the selection of the Census Transform [1], [2] for implementation. The Census Transform is in the class of area-based algorithms and is well-suited for acceleration in FPGAs, as the majority of the operations performed consist of bit manipulations and simple arithmetic. The Census Transform also exhibits relative insensitivity to absolute differences in imaging devices, and operates well on images with high dynamic range.

The Census Transform begins by considering a window around the pixel under test from one (the primary) image. An example of a window with radius "1" is the following, centered on a pixel with intensity 130:

$$\begin{array}{ccc} 127 & 129 & 131 \\ 126 & 130 & 129 \\ 126 & 131 & 133 \end{array}$$

This window of intensities is then transformed into a bit string by computing whether each of the neighbor pixels in the window is greater or less than the center value. For this example, pixels with lower intensities are given a value of 1, and pixels with higher intensities are given a value of 0. Pixels with intensity values equal to the intensity value of the center pixel can be treated as either a 1 or 0, but must remain consistent. The transformed matrix would therefore be:

$$\begin{array}{ccc} 1 & 1 & 0 \\ 1 & X & 1 \\ 1 & 0 & 0 \end{array}$$

This matrix is then rearranged into a bit string to capture the entire window, resulting in the eight-symbol pattern: "11011100". With this pattern that represents the window about the pixel under test in the primary image, we must find the closest matching pixel window in the secondary image by calculating the corresponding bit strings for several possible matching center pixels in the secondary image and comparing bit strings. For our implementation, we used the Hamming distance between the bit strings as a comparison measurement. The Hamming distance is simply the number of places in the bit string where bits in the same position differ. Pixel windows with similar intensities relative to the center pixel will have similar bit strings, which in turn will yield smaller Hamming distances.

The same pixel position in the secondary image yields a disparity of zero and an infinite range estimate. The algorithm continues to shift and compare until a predetermined maximum disparity is reached, at which point the search is stopped. Now, among the set of Hamming distances, the smallest corresponds to the pixel under test's most likely disparity value.

D. Parameterization

The need for parameterization can be most acutely demonstrated during prototyping of robotic platforms. In deciding and evolving platform characteristics such as the physical camera separation (baseline), the camera type (possibly changing pixel depth and resolution), the physical environment (indoor vs. outdoor), the expected speed of the robot, and the available FPGA resources, basic algorithmic parameters require constant manipulation to achieve the desired results. Making these changes requires non-trivial modification to the hardware design of the stereo vision core.

By being able to quickly generate the hardware stereo core, it becomes possible to perform experiments in situ, in hardware, with little delay. Additionally, by storing a large number of configurations, platforms can now determine and utilize, in the field, the best parameters for operating conditions.

E. Related Work

As many contemporary stereo vision algorithms have heavy computational requirements, a number of acceleration efforts have been documented to date. Of interest are FPGA-based implementations [8]–[10], and dedicated commercial ASIC solutions (Tyzx Deep Sea products), none of which offer the parameterization of our implementation.

Algorithmically, the most closely related work is [8]. The Census Transform was used here on the PARTS Reconfigurable Computer, consisting of 16 Xilinx 4025 FPGAs and 16MB of SRAM. Our implementation is a generalization of this basic idea to be able to target both low-cost, small FPGAs, and large, high-performance FPGAs, while obviating the need for external memories.

In [9], [10], the authors implement phase-based correspondence algorithms that differ significantly from the Census Transform in both design and arithmetic complexity. These implementations, however, are single instances and do not allow for easy parameter tuning.

Stereo vision remains an active area of research whose breadth and depth is beyond the scope of this paper. Interested readers are encouraged to explore [7] for an a broad survey of the field of study and a quantitative and qualitative comparison of approaches.

II. Implementation

The implementation of the stereo vision core was done in Verilog, simulated with ModelSim, and synthesized using the Xilinx toolchain. Parameterization was accomplished using a combination of basic Verilog `generate` constructs and Verilog code generation with software written in Python. The parameters that are available to the end-user to modify include:

- pixel depth
- image width
- image height
- search window size
- maximum disparity

The *pixel depth* parameter adjusts the maximum dynamic range. Typically, this would be 8 bits per pixel. As the implementation also generates a simulation model for typical OmniVision-brand CMOS imagers (as previously used in [3]), the *image width* and *image height* parameters are used for the simulation of row and frame timing information. The *search*

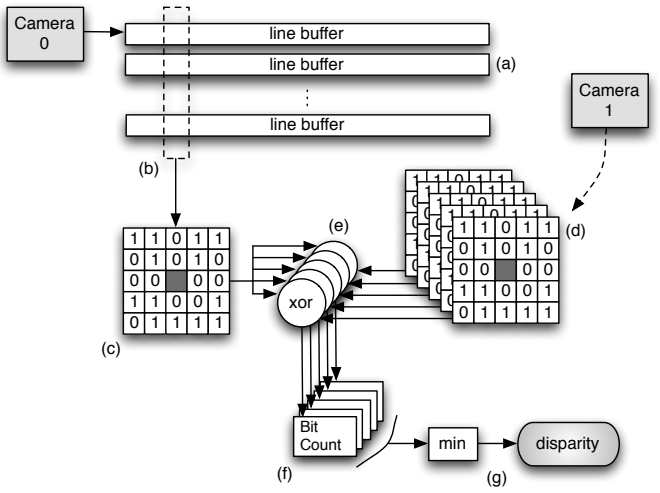

Fig. 1. Graphical representation of data flow in the Census Transform.

window size parameter specifies both the width and height of the square window of pixels surrounding the pixel under test. Finally, the *maximum disparity* parameter specifies the amount of lateral shift that will be searched to find correspondence in the secondary image.

Figure 1 depicts the flow of data from cameras to disparity calculation and closely mirrors the previous description. Both *Camera 0* and *Camera 1* are assumed to operate synchronously—that is, they deliver the same pixel coordinate at the same time, and have synchronized pixel clocks. This implementation is streaming in nature, and will deliver a disparity calculation at every pixel clock with some fixed latency. Starting from *Camera 0*, a pixel is generated with every pixel clock.

Pixels must be stored in *line buffers* **(b)** until there are enough pixels for a window to be formed around the pixel under consideration. A single line buffer width is the same as the image width, and we require *search window size* number of them. The Xilinx synthesizer is able to utilize BlockRAM to implement these line buffers efficiently.

Once enough pixels are available to begin filling a window around the pixel under consideration, a register bank of $(search\ window\ size)^2$ registers **(c)** is filled, a column at a time, from the line buffers. At every column fill, the binary relative intensity is calculated and stored in a register array the same size as **(c)** (not shown).

At the same time, *Camera 1* has been delivering pixels in an identical fashion to a different instance of the same structure of line buffers and pixel window registers **(d)** (not shown, and indicated by the dashed line). The only difference is that there are *maximum disparity* copies of the pixel window registers. As soon as there are enough pixel windows available from *Camera 1* to compare against the pixel window from *Camera 0*, the current pixel window from *Camera 0* is XOR'ed against all *maximum disparity* number of pixel windows from *Camera 1* **(e)**.

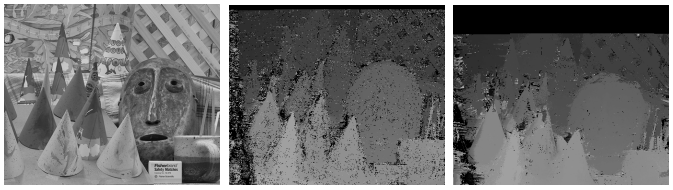

Fig. 2. "Cones" from [11]: (left) original image; (center) output, window size 10x10, maximum disparity 90; (right) output, window size 60x60, maximum disparity 90.

The number of differences resulting from these parallel XORs are calculated in **(f)**, giving a scalar value that represents the level of mismatch between the pairs of images. The index of the minimum of these values is then taken **(g)** and delivered as the resultant disparity estimate at the position of the pixel under consideration from *Camera 0*.

A. Simulation and Verification

The design was simulated and verified using ModelSim SE 6.4c with a variety of parameter configurations using images from [11], [12]. Simulation output demonstrating the effects of parameter modification can be seen as raw output in Fig. 2. No pre- or post-processing that might typically accompany range estimation has been done except to scale disparity values to occupy the full gray scale.

Comparing the middle and right images of Fig. 2, one can easily see the smoothing impact of an increased window size. Less noise is present, while less detail is also found. The right image is vertically offset as the algorithm does not currently begin processing until there is a full window for evaluation. Therefore, going from 10 to 60 pixels in the window increases the "band" at the top of the image by 50 pixels.

B. Parameter Evaluation

The design was synthesized for a variety of parameters using the Xilinx ISE 9.2.04i. The target FPGA was a Xilinx Spartan-3 XC3S2000, chosen to match [3]. In Fig. 3, we can see some reasonable trends in logic utilization versus maximum disparity for a variety of pixel window sizes.

Fig. 3 utilizes an image size of 320x240, pixel depth of 8 bits, maximum disparities from 2 to 40 pixels, and window sizes from 7 to 19 pixels. As the maximum disparity increases, the number of registers required to store pixel windows increases. The number of XORs required to compute the mismatch between pixel windows increases with maximum disparity, and the size of each XOR scales with the size of the pixel window. Additionally, the number of bit counters required to calculate the scalar correspondence between pixel windows increases with maximum disparity, and the complexity scales with the size of the pixel window. Therefore, an increase in either maximum disparity or window size should demand more resources. For large window sizes and large maximum disparity, the number of data points is fewer than others as the resource requirements exceeded the capacity of the target FPGA. BlockRAM usage is not shown,

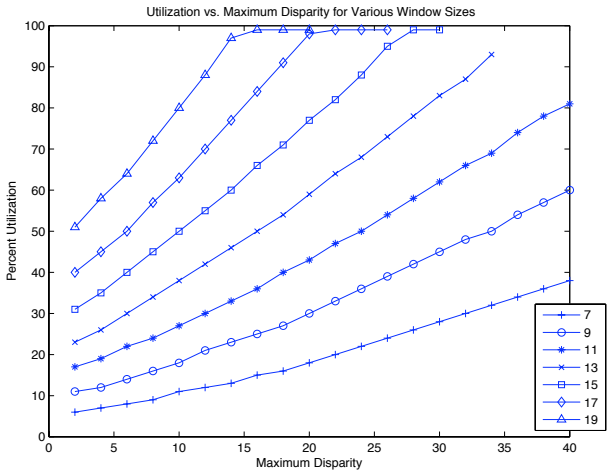

Fig. 3. FPGA logic utilization versus maximum disparity for various window sizes. Image: 320x240, pixel depth: 8 bits, disparities (2-40), window size (7-19).

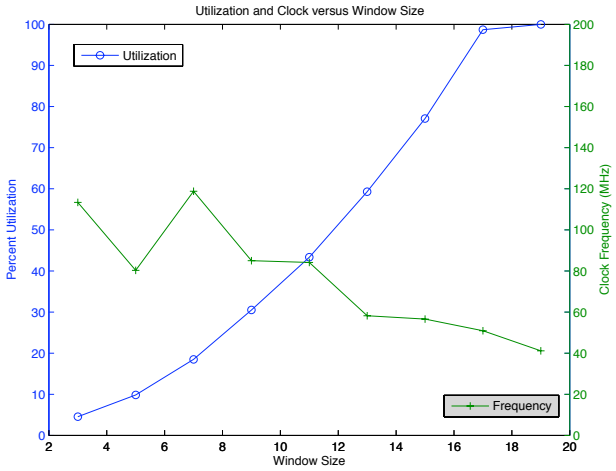

Fig. 4. FPGA utilization and clock frequency versus window size for maximum disparity of 20. Image: 320x240, pixel depth: 8 bits, window size (3-13).

however, it scales similarly to logic and is dependent primarily upon window size.

Finally, it is noteworthy to compare our results to our previous implementation [3]. With the same parameters, our current implementation uses nearly exactly the same logic resources (59% versus 57%), the same amount of BlockRAM (26 out of 40), and runs at 58MHz instead of 26MHz. The frequency discrepancy is due to the fact that our previous design included support logic to drive the USB host interface. At 58MHz, the current design can support over 300 320x240 grayscale frames per second.

To get a more detailed notion of trends, a single maximum disparity of 20 pixels is chosen for Fig. 4, which plots FPGA utilization and clock frequency versus window size between 3 and 13 pixels square. Again, the trends are as expected, with utilization scaling with window size, and the clock frequency trending downward, although with significant noise attributable to the probabilistic nature of the place and route processes.

Finally, run times for each implementation with the Xilinx suite of tools on a 3.4GHz Intel Xeon workstation were between 5 and 30 minutes, depending on utilization. Code generation completes in under a second.

III. Conclusions and Future Work

We have presented our work in developing a parameterized stereo vision core which allows for the rapid exploration of the design space of stereo vision systems. Our implementation functions as expected, and can dramatically decrease the time necessary to implement stereo vision systems.

We are currently developing area and performance models to facilitate accurate estimation of resource utilization so users may make more informed design decisions without needing to perform an implementation. We are also expanding our code generator to implement a suite of stereo vision algorithms that include optical flow, phase-based, and other approaches, allowing the user to explore an even wider range of solutions with ease.

References

[1] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *ECCV '94: Proceedings of the third European conference on Computer Vision (Vol. II)*, 1994, pp. 150–158.

[2] R. Zabih, "Individuating unknown objects by combining motion and stereo," Ph.D. dissertation, Stanford University, 1994.

[3] C. Murphy, D. Lindquist, A. M. Rynning, T. Cecil, S. Leavitt, and M. L. Chang, "Low-cost stereo vision on an FPGA," in *Proc. 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines FCCM 2007*, 23–25 April 2007, pp. 333–334.

[4] R. D. Henkel, "Fast stereovision by coherence detection," in *CAIP '97: Proceedings of the 7th International Conference on Computer Analysis of Images and Patterns*. London, UK: Springer-Verlag, 1997, pp. 297–304.

[5] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 993–1008, 2003.

[6] K. Mühlmann, D. Maier, J. Hesser, and R. Männer, "Calculating dense disparity maps from color stereo images, an efficient implementation," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 79–88, 2002.

[7] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[8] J. Woodfill and B. V. Herzen, "Real-time stereo vision on the parts reconfigurable computer," in *IEEE Symposium on FPGAs for Custom Computing Machines*. IEEE, April 1997.

[9] A. Darabiha, J. Rose, and J. W. Maclean, "Video-rate stereo depth measurement on programmable hardware," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 18–20 June 2003, pp. I–203–I–210.

[10] D. K. Masrani and W. J. MacLean, "Expanding disparity range in an FPGA stereo system while keeping resource utilization low," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 25–25 June 2005, pp. 132–132.

[11] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 18–20 June 2003, pp. I–195–I–202.

[12] C.-C. Wang. Vision and autonomous systems center's image database. Carnegie Mellon University. [Online]. Available: http://vasc.ri.cmu.edu/idb/