Intensity and Distance Thresholding in Hardware to Enable Flexible Blob Detection for a Vision System with Limited Bandwidth

Peter Samarin, Timur Saitov, Kenneth B. Kent, Rainer Herpers

This work presents an approach to blob detection that divides the processing between an FPGA and a PC and balances the precision of blob detection with the bandwidth usage. The FPGA receives images in a raster-scan order from an on-board camera and separates foreground pixels from the background by using two thresholds—an intensity threshold and a distance threshold. The foreground pixels are transferred to the PC by an Ethernet interface.

The PC performs blob detection on received images by using connected component labeling and calculates the centers of mass of each detected blob. Evaluation results show that for a camera that captures 8-bit grayscale images with resolution of 640x480 pixels, running at 100 frames per second and a bandwidth limited to 50 Mbit/s, the FPGA is able to transmit a sufficiently large number of blobs that allows for high precision blob detection to be performed on the PC in real time.

1 Introduction

Detection and analysis of binary large objects (blobs) is an important problem in computer vision, and is often used for image segmentation and extraction of features from objects of interest. Blob detection is used in our virtual reality environment—the Immersion Square (Herpers et al., 2005)—as an intermediate step in estimating position and orientation of an interaction device (Bochem et al., 2010).

In the current version of the Immersion Square, the user is surrounded by three projection screens and can interact with the system by using a mobile 6 degrees of freedom interaction device. Three projectors cast a set of infrared light spots arranged in a specific pattern onto the screens from the back. The interaction device is composed of a camera and an FPGA that transmits data to a PC over a wireless interface. User input is generated based on the 3D position and orientation (pose) of the device, which is computed by detecting the centers of bright spots and by computing the pose that best matches the pattern (Scherfgen et al., 2011).

It has been shown in previous research that for 3D pose estimation out of blob patterns, high precision in the sub-pixel range of the blob detection process is mandatory (Bochem et al., 2010). For that, high level image processing approaches need to be applied, which, however,

cause performance and implementation conflicts on FPGA side. In (Bochem et al., 2010), the computation of centers of the bright spots was performed completely on the FPGA. However, the results were not accurate enough for subsequent camera pose estimation because for bright spots with smooth borders caused by a real light scattering and camera chip smoothing, approaches only using intensity thresholding produce sharp cutting borders. These borders oscillate due to the absence of a hysteresis and cause notable changes in calculation of the center of mass of each blob.

In this contribution, a hybrid approach to blob detection for our system with limited bandwidth is presented, where an FPGA is used to preprocess the images by extracting the foreground pixels of roughly detected blobs and sending them to the PC. The foreground pixels are separated from the background pixels by using an intensity threshold in combination with a distance threshold. On PC side, the centers of the bright spots are computed in subpixel precision and are subsequently used to estimate the 3D camera pose. This approach is implemented and evaluated on a prototype with a bandwidth limited to 100 Mbit/s.

2 Approach

The high level view of the approach is shown in Figure 1. Camera images arrive in a raster scan order at the FPGA. Small parts of the image are stored in a partial frame buffer for later lookup. The foreground pixels are separated from the background pixels by using two thresholds. The first threshold compares the intensities of pixels with a predefined value and discards pixels whose intensities are lower. Pixels whose intensities exceed the threshold are considered as foreground pixels. The second threshold extends the borders of the blob by adding pixels within a fixed range of each blob. A bitmap is used to temporarily keep the status of each pixel—whether a pixels belongs to foreground or to the background. The bitmap is used in a later step to look up the intensities of each foreground pixel in the partial frame buffer. The intensities and their coordinates are run-length encoded and sent to the PC by following a custom data transfer protocol that is based on the Ethernet protocol. The PC performs blob analysis on received image regions and extracts features from each blob.

2.1 Foreground Pixel Recognition on FPGA

2.1.1 Intensity and Distance Thresholding

When the intensity threshold T_I is used to separate foreground pixels from the background, all pixels whose intensities exceed a predefined threshold are retained and all other pixels are discarded. The upper part of Figure 2(a) shows a magnified image of a bright spot generated by a laser directed at the screen of our virtual reality environment. The lower part of Figure 2(a) plots the corresponding intensities as elevation along the Z-axis. Thresholding with the intensity threshold of 10, as shown in Figure 2(b), corresponds to drawing a plane that is parallel to the XY plane at intensity equal to 10.

One of the challenges of using the intensity threshold is to define a value that allows accurate feature extraction without exceeding the available bandwidth. If the threshold is set too high, it can cut off important pixels on the edge of the bright spots and negatively impact the precision of further processing. Conversely, if the threshold is set too low, many pixels will be considered as foreground pixels, which will result in high bandwidth usage.

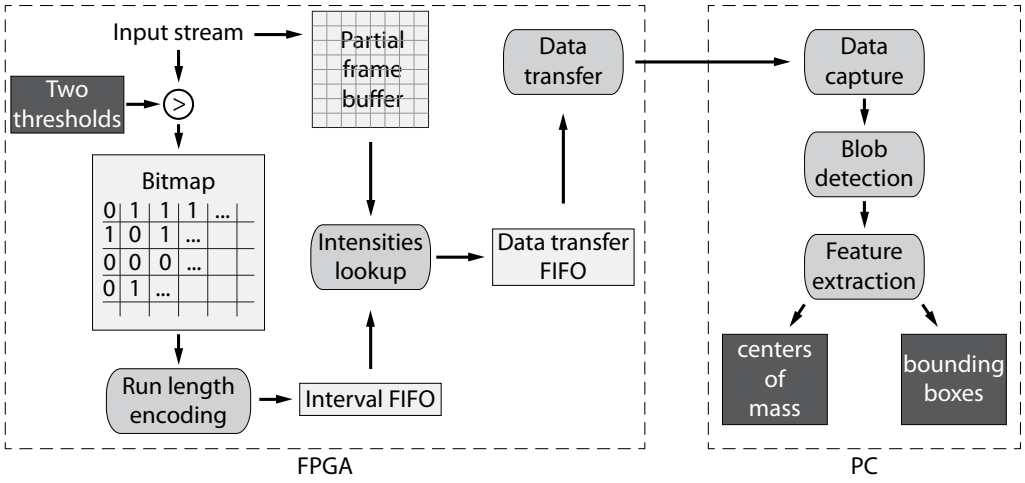

Figure 1: System design. Processing is split up between FPGA and PC. On FPGA side two thresholds are applied onto camera images and transfers foreground image parts to the PC. Connected component labeling as well as the extraction of features of the foreground pixels is computed on PC side.

To overcome this problem, a second threshold called the distance threshold T_D extends the blob border obtained from applying the intensity threshold. The distance threshold is based on *chessboard distance* (Borgefors, 1986). The chessboard distance between two pixels is defined as: $D(x1, y1, x2, y2) = \max(|x_1 - x_2|, |y_1 - y_2|)$. The underlying reason of this approach is that a Gaussian distribution of the blob shape and therefore a point-symmetrical blob shape on image level is assumed.

When the distance threshold T_D is used in combination with the intensity threshold T_I, all pixel that are within a fixed chessboard distance T_D from the foreground pixels that pass the intensity threshold, are also considered foreground pixels. Figure 2(d) shows the result of applying both thresholds. By using two thresholds, the intensity threshold can be set to a relatively high value, while the distance threshold allows control over the bandwidth usage by extending or shrinking the blob boundaries by a fixed distance.

2.1.2 Thresholding Implementation

In case when only the intensity threshold is used, the foreground pixels can be sent directly to the PC. However, when the distance threshold is applied, further background pixels might have to be added after processing several lines. Hence, all foreground pixels must be temporarily stored before they are sent to the PC.

Our approach implements the two thresholds by using a bitmap that holds a binary version of several lines of the camera image after the intensity threshold has been applied. The bitmap only needs one bit for storing the pixel status ('0' denotes background pixels, '1' denotes foreground pixels). The algorithm 1 shows how thresholding with the intensity threshold T_I and the distance thresholds T_D is performed. Line 1 applies intensity thresholding to pixel $P(x, y)$. Lines 2 and 3 loop over all the pixels that are within the distance threshold T_D from P.

The bitmap is implemented in hardware by using registers that allow random access and

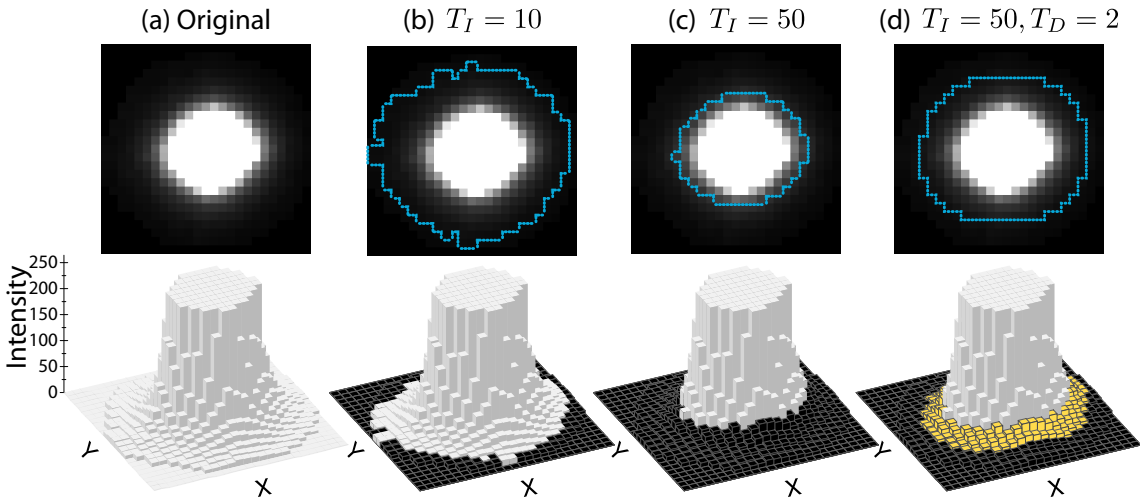

Figure 2: Intensity and distance thresholding approach. (a) Magnified image (top), and the same image with intensities plotted as elevation along the Z-axis (bottom). (b) Application of the intensity threshold of $T_I = 10$ results in a border that is shown by the dashed blue line surrounding the bright spot. Pixels inside this border are considered as foreground pixels, while pixels outside of the border are background. (c) Application of the intensity threshold $T_I = 50$. (d) Using intensity threshold $T_I = 50$ in combination with the distance threshold $T_D = 2$. Pixels added by T_D are marked by yellow color.

Input: Pixel intensity $I[x; y]$, pixel coordinates x and y

```
1 if I[x;y] > T_I
2     for x_D = −T_D to T_D
3         for y_D = −T_D to T_D
4             if not(pixelOutOfBounds(x, y, x_D, y_D))
5                 y_bitmap ⟵ mapImageToBitmap(y, y_D)
6                 bitmap [x + x_D; y_bitmap] ⟵ '1'
```

Algorithm 1: Implementation of intensity and distance thresholding on FPGA.

can be read and written in the same clock cycle. Every time when a new foreground pixel is detected by the intensity threshold, MxM bits are set to '1' in the bitmap, where $M = 2 * T_D + 1$. The MxM image region includes all pixels within the chessboard distance of T_D surrounding the foreground pixel.

2.2 Data Transfer

After applying thresholding to an image line y, the foreground pixels in line $y - T_D$ are transmitted to the PC. The bitmap is used to look up the status of each pixel (foreground or background), and the partial frame buffer to look up the intensities of the foreground pixels. The intensities and the corresponding coordinates of the foreground are stored in the data transfer FIFO. In order to save space in the FIFO, run-length encoding is applied.

The PC must be able to distinguish packets with foreground pixels from all other network traffic. Therefore, a custom protocol that is based on the Ethernet protocol has been developed to send regions of interest to the PC. Each packet consists of a header and a payload. The header is comprised of the MAC addresses of the FPGA board and the PC. The payload contains the run-length encoded pixel intensities. Each run is defined by its line coordinate in the image (y), the start and the end of the run, as well as the corresponding pixel intensities.

2.3 Blob Analysis

A generic approach to blob detection on the PC is realized by applying a specially adapted connected component labeling (CCL) method (Rosenfeld and Pfaltz, 1966). The idea of CCL is to examine every pixel that exceeds a fixed threshold and to determine whether any of its 8 neighbors also exceed the threshold. Such pixels are grouped together. Since most of the pixels received from the FPGA are foreground pixels, connected component labeling takes less time than in cases where a full image has to be processed.

Features such as the bounding box and the center of mass are extracted on the fly as described in (Bailey, 2011). The bounding box for pixels with the same label is computed by constantly updating the minimum and maximum of X and Y coordinates of all connected components. The center of mass for blob is computed by summing up the coordinates of each pixel multiplied by their respective intensity values and normalized by the accumulated pixel values.

3 Experimental Setup

The approach developed in this contribution has been implemented by using VHDL on Altera's DE2-70 board developed by Terasic. The core of the board is a Cyclone II 2C70 FPGA device with 68,416 logic elements and 250 M4K RAM blocks. The FPGA is connected to several interfaces that are used for communication with the PC, such as a serial port and a 100 Mbit/s Ethernet port. A mvBlueCOUGAR-X 100 camera (Lansche, 2012) is mounted on one of two general-purpose expansion headers (GPIO) of the board. The camera has a CMOS sensor that delivers 10-bit grayscale images with resolution of up to 752x480 pixels and allows frame rates of up to 117 frames per second. The camera has an on-board FPGA and can be configured from a PC over a 1 Gbit/s Ethernet connection.

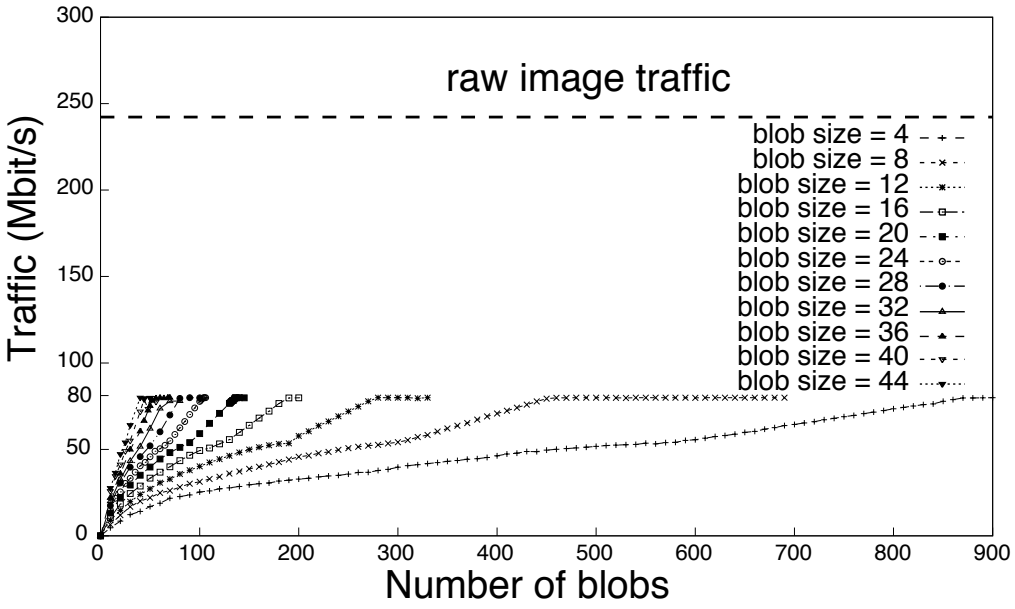

Figure 3: Average traffic for blob sizes of 4x4 to 44x44 with step size of 4. All images have sizes of 640x480 pixels with 8-bit grayscale values. Distance threshold T_D is set to 2. For comparison, the dashed horizontal line shows the bandwidth average bandwidth usage of the camera running at 100fps.

4 Evaluation

The evaluation of the thresholding approach has been performed by measuring the network traffic generated by the system under different loads. In order to evaluate the approach in a systematic way, the number of foreground pixels must be controlled. However, it is impossible to precisely control their number when using images of real cameras because of noise and other influences. This problem can be overcome by using synthetic images instead. For this purpose, a dataset of 2000 random synthetic images has been generated. The images vary in number of blobs and size.

Figure 3 shows the dependency between the number of blobs, selected blob sizes and the network traffic that results from sending run-length encoded regions to the PC. The graph shows that a large number of blobs can be transferred to the PC as long as their size is small. The camera can run at high frame rates while an accurate blob detection algorithm is performed on the PC.

Figure 4 shows the usage of logic elements (LEs) dependent on the distance threshold. Circuits with distance thresholds ranging from 0 to 30 have been synthesized by using Quartus. The size of the circuit depends linearly on the distance threshold. No embedded multipliers are used by the design. Table 1 shows the block RAM usage of different modules.

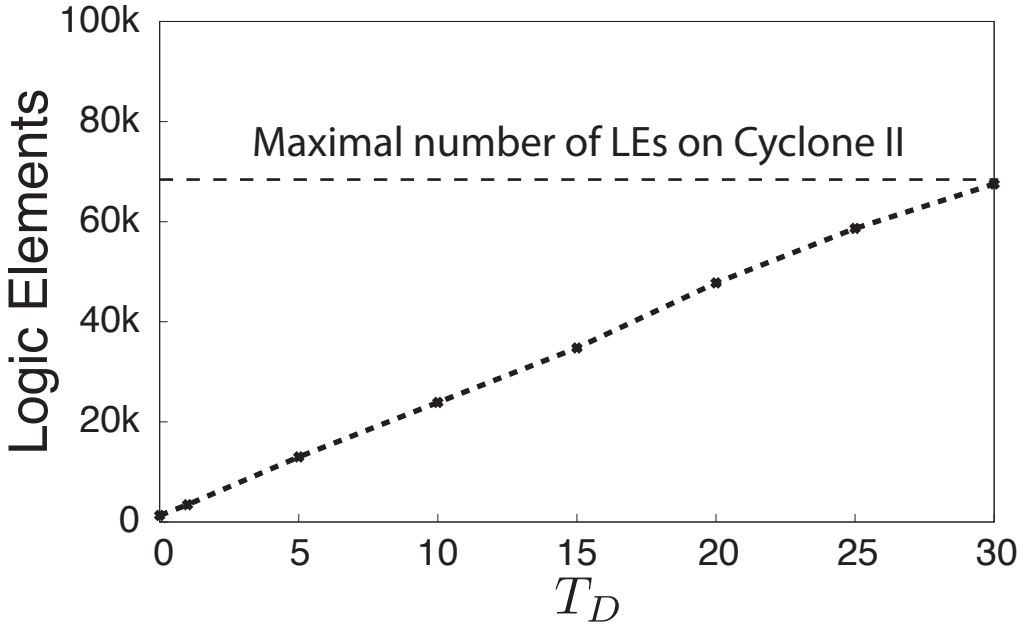

Figure 4: Usage of logic elements (LEs).

Table 1: Dedicated block RAM usage.

Module	Memory bits
Interval FIFO	94,208
Partial frame buffer	400,000
Data transfer FIFO	524,288
Total	1,020,544

5 Conclusions and Future Work

The approach to intensity and distance thresholding presented in this contribution has several advantages compared to simple intensity thresholding for images containing dots with smooth borders caused by real light scattering. If the intensity threshold is used alone, it might cut off important blob areas, or include too many unimportant pixels and overload the available bandwidth. However, if it is used in combination with the distance threshold, it allows the intensity threshold to be set to a relatively high value and still be able to capture important pixels. The distance threshold extends the border of the blobs by a given distance, which provides a way to control the bandwidth usage by trading it off against precision of subsequent feature extraction.

The main advantage of using intensity and distance thresholding on FPGA is that it allows high precision blob analysis to be performed on the PC. This saves development time of a complex algorithm on FPGA side. Implementing an algorithm on an FPGAs is inherently harder than implementing the same algorithm on PC. Another advantage is the flexibility offered by the general purpose processor—the program performing blob analysis can be adapted in order to meet the requirements of the virtual reality environment. This saves development cost and

shortens time to market. Yet another advantage of the PC is the library support for computer vision applications—for example there are several blob detection algorithms available in the OpenCV library (Bradski, 2000), whereas no such library exists for FPGAs.

Several directions for future work can be outlined. Currently, the bitmap that is used for temporally storing foreground pixels is implemented by using registers. This has the disadvantage that a large number of logic elements is used even when the distance threshold is relatively low (1 to 5). However, by saving the bitmap in block RAM, it is possible to reduce the usage of logic elements. This will result in circuits that can fit into smaller FPGAs, or use these logic elements to perform other tasks.

Another direction for future work is to balance the available bandwidth and precision of blob detection by adding a blob analysis circuit to the FPGA and by using it in parallel with the blob analysis approach on the PC. Under high loads, the system on FPGA side can decide to transmit approximately computed blob features instead of blob regions, which will reduce bandwidth usage at the cost of precision.

Acknowledgments

The authors gratefully acknowledge the financial support of the BMBF in the FHprofUnt program line; project 6-MIG grant No: 1759X08 and the DAAD PPP project, No 50750255, "FPGA based Computer and Machine Vision". The authors would like to thank CMC Microsystems as the tool provider and the Natural Sciences and Engineering Research Council of Canada for their contributions to this project.

Bibliography

Bailey, D. (2011). *Blob Detection and Labelling*, chapter 11, pages 343–375. Wiley-IEEE Press, 1st edition.

Bochem, A., Herpers, R., and Kent, K. (2010). Hardware acceleration of blob detection for image processing. In *Third Int. Conf. on Advances in Circuits, Electronics and Micro-Electronics (CENICS)*, pages 28–33.

Borgefors, G. (1986). Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344 – 371.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Herpers, R., Hetmann, F., Hau, A., and Heiden, W. (2005). The Immersion Square – A mobile platform for immersive visualizations. In *Aktuelle Methoden der Laser- und Medizinphysik: 2. Remagener Physiktage 2004*, pages 54–59.

Lansche, U. (2012). *mvBlueCOUGAR-X documentation, V1.0b24*.

Rosenfeld, A. and Pfaltz, J. L. (1966). Sequential operations in digital picture processing. *J. ACM*, 13:471–494.

Scherfgen, D., Saitov, T., Herpers, R., and Dayangac, E. (2011). An optical laser-based user interaction system for cave-type virtual reality environments. In *Proc. of the 4th Russian-German Workshop "Innovation Information Technologies: Theory and Practice"*.