Binary Histogram based Split/Merge Object Detection using FPGAs

Kofi Appiah, Hongying Meng, Andrew Hunter, Patrick Dickinson
Lincoln School of Computer Science, University of Lincoln
Brayford Campus, Lincoln
{kappiah, hmeng, ahunter, pdickinson}@lincoln.ac.uk

Abstract

Tracking of objects using colour histograms has proven successful in various visual surveillance systems. Such systems rely heavily on similarity matrices to compare the appearance of targets in successive frames. The computational cost of the similarity matrix is increased if proximate objects merge into a single object or a single object fragments into two or more parts. This paper presents a method of reducing this computational cost with the use of a reconfigurable computing architecture. Colour histogram data of moving targets are used to generate binary signatures for the detection of merged or fragmented objects. The main contribution in this paper is how binary histogram data is generated and used to detect split/merge object with the use of logical operations native to the hardware architecture used for its implementation. The results show a 10 fold improvement in processing speed over the microprocessor based implementation, and that it is also capable of detecting split/merge objects efficiently.

1. Introduction

Visual surveillance systems usually have the ability to detect moving objects, track them in the field of view and perform high level behavioural analysis [1]. When the camera is stationary, a natural approach to detect moving objects is to use background differencing, in which a reference image is maintained and constantly updated to reflect changes in the scene. Images acquired from the camera are compared with the reference image to extract any dissimilarity, which is further processed into objects.

Moving objects detected are then tracked from frame to frame by establishing correspondence between persistent blobs in the scene. Tracking is a very important intermediate processing step in visual surveillance, yet it is susceptible to segmentation errors. Most of the problems arise when two moving objects gets close to each other, when an object become partially occluded, when there is a new object in the scene and when an existing object leaves the scene. Yilmaz *et al.* [20] categorises tracking algorithms into three major groups: point-based [4], kernel-based and silhouette-based [8]. Wren *et al.* [18] presents a silhouette-based approach, which uses object colour and location statistics to generate the trajectory of an object in every frame. Boykov *et al.* [3] presents a point-based approach using a Kalman filter. The system tracks rigid objects based on an adaptive recognition technique that incorporates dependencies between object features. In [7], Collins presents a kernel-based tracking system using the mean-shift algorithm. The problem associated with the choice of correct scale for tracking a blob using the mean-shift algorithm has also been addressed using difference of Gaussian.

An ideal visual tracking system should be temporally and spatially complete [2]. Thus, there should be no single frame in which an object in the field of view would not be accounted for – temporal completeness; and each object in the field of view should correspond to a single connected silhouette (blob) in the image space that completely covers only the object's pixels – spatial completeness. In an environment with a variable number of objects from different classes, it is not always possible to achieve both temporal and spatial completeness during tracking.

Owens *et al.* [14] presents a model-free tracking algorithm which is capable of handling object grouping and fragmentations. The system uses area and histogram of the object pixels to match objects to silhouette from frame to frame. A cost minimization function is employed to merge fragmented silhouettes. Similarly, a silhouette regression line histogram is used to partition merged silhouettes. The paper reported 84.9% accuracy in tracking multiple objects over a period of three days.

The main challenge with object tracking is also addressed by Bose *et al.* [2] using an inference graph. Using the inference graph and a generic object model, tracked object blobs are labelled as either whole object, fragments of objects, or groups of interacting objects. The model uses the average difference in speed between any pair of elementary targets. The algorithm takes approximately 40s in tracking multiple objects recorded over a period of 30s.

Takala *et al.* [15] presents a colour, texture and motion based tracking systems capable of running in real-time. The system used RGB colour histogram and correlogram to describe the object's colour properties, and local binary patterns for texture and geometric location to handle merging and splitting of objects. The system has reported under performance when object are emerging in the camera scene. It is capable of tracking multiple objects in diverse conditions while achieving speeds of $10 - 15$ frames per second on a 3GHz computer.

Ling *et al.* [11], emphasized the ability of using colour features to achieve robust object tracking. The system extracts colour information cluster-by-cluster to compare the object's similarity across the image sequences. Clusters are derived for each moving object based on the number of areas with similar RGB colour properties. Weighted cluster-based matching is then used to compare cluster colour information between the current and previous frame. Results show that the cluster colour information can be used in detecting merged objects as well as fragmented objects.

Withagen *et al.* [17], also presents a colour histogram based object detection and tracking system using a likelihood-based framework to classify object and background pixels. The likelihood framework starts with matching of histogram data to find the best location of the object-core followed by the classification of pixels into objects. Johnsen *et al.* [10] presents and object tracking and classification systems capable of tracking multiple objects in real-time. To successfully classify an object as a person, group of people, or a vehicle, the assumption that objects merge to form a single object of size similar to the sum of the two objects is made. The systems works in real time with reported processing speed of about 50 frame per second for a 768×576 size colour image.

The need to handle fragmentation and grouping of moving object silhouettes in a robust visual tracking system has been mentioned above. Typically, a similarity matrix is use to establish a frame-to-frame correspondence between objects and silhouettes. The computational requirement increases significantly if the number of objects is varying and unknown. This paper presents an object tracking component, implemented on a Field Programmable Gate Array (FPGA) to enhance the real-time multiple object tracking in a visual surveillance system.

The paper is organised as follows, section 2 presents related FPGA object tracking systems. Section 3 gives a general description of the entire system and where the module presented here fits into a complete tracking system. Section 4 gives the theory alongside the rules applied to resolve fragmented and grouped silhouettes using binary histogram data. This is followed by the FPGA architectural detials of the design in section 5. Sections 6 and 7 present some experimental results and conclusion respectively.

2. Related Work

Object detection and tracking algorithms need to run in real-time, if they are to be used for the purposes of security surveillance. Most robust tracking systems are implemented purely in software, exhibiting a high level of robustness with little or no real-time processing capabilities. Muhlbauer *et al.* [13] presents the design and implementation of a feature tracking systems on reconfigurable hardware. The design is a modular implementation of a feature tracker on a Xilinx Spartan3 FPGA, capable of processing VGA size image at $162fps$. McErlean [12], presents a motion detection and object tracking system targeting an Altera Stratix FPGA. The design is based on an algorithm developed specifically for hardware implementation, capable of detecting and tracking moving objects. The design of the system allows for a number of Block Based Phase Correlation pipelines to be instantiated in parallel. The use of FPGA BlockRAM as pipeline buffers in the system implementation has contributed to the increased performance.

Xu *et al.* [19] presents an FPGA implementation of the Active Shape Model algorithm for recognition of non-rigid objects. The design takes advantage of the parallel nature of the algorithm by implementing a deep pipeline structure for its acceleration. The design has been accomplished on a single Altera Stratix FPGA with a speed up of about 15 times that of a software implementation running on an Intel Pentium 4 processor.

Cho *et al.* [6], also presents a real-time visual tracking system fully implemented on FPGA using colour histograms. The proposed visual tracking circuit searches all regions of the image to perform a matching operation in order to estimate the position of a moving object. The successful implementation of the tracking circuitry on FPGA has a speedup of approximately 9 times compared to a software implementation on an Intel Pentium 4 processor.

Wang *et al.* [16] presents a real-time object tracking systems implemented on TMS320DM643 DSP and a Spartan3E FPGA. The design takes advantage of the flexibility of DSP and the logic rich properties of the FPGA architecture, and can be used for tracking short-range air-defence anti-missile weapon.

Point-based tracking methods like Kalman filtering, particle filtering and condensation algorithms have successfully been used for tracking multiple objects [6, 20]. However, they fail to track successfully when used to track fast moving objects. Silhouette-based approaches, which relies on colour histograms can be used to track fast moving objects because of their robustness to rotation, partial occlusion and non rigidity of the targets [9]. Similarly, the silhouette-based approach can fail due to changes in illumination in camera parameters or in object motion [6].

To demonstrate the potential of colour information in handling split/merge silhouettes on a hardware architecture for the purposes of security surveillance, binary signatures extracted from colour histogram of moving objects are used to establish when two objects group into a single silhouette, and when a single object fragments into two silhouettes.

3. System Overview

A system capable of detecting when two tracked objects merge into a single object, or when a single tracked object fragment into parts, has been developed and tested. Various issues may contribute to these two problems (split and merge) in tracking. When a tracked object is partially occluded by a stationary object like a tree or a fence, the object silhouette may fragment into parts. Similarly, when part of a moving or tracked object has very similar colour and texture properties to the background, pixels of the tracked object can be misclassified, which can also cause fragmentation. Not only can fragmentation cause problems during tracking, but also in the grouping of objects. Spatially proximate objects can cause tracked objects to merge temporarily into a single silhouette [2].

Most silhouette-based tracking algorithms like the one presented by Owens *et al.* [14] are purely measurements based object-to-silhouette matching, which attempt to track objects in that form they are initially detected. Therefore, the system always attempts to track distinct objects even if their silhouettes merge with that of other objects. A best-match process (similarity matrix), a computationally intensive task, is used to match silhouettes conservatively to existing objects from frame-to-frame. The task of matching silhouettes to objects becomes more computationally demanding when the number of objects varies from the number of silhouettes in the current frame.

This is usually the case when spatially proximate objects merge or objects leave the field of camera view – thus the total number of objects will exceed the total number of silhouettes. Similarly, when a partially occluded object is fragmented into parts or when a new object enters the field of camera view, the total number of objects will exceed the total number of silhouettes. Our goal is to use a massively parallel architecture, like FPGA, to speedup the best-match process in a silhouette-based tracking algorithm. The role of our system is to determine when a new object has entered the scene as opposed to a single object fragmentation. Also the system should determine when an object leaves the scene as opposed to grouping by proximity.

To achieve this challenging task on FPGA architecture, taking advantage of data parallelism and deep pipelining, binary signatures extracted from colour histograms of moving objects have been used. The FPGA is used as a coprocessor to facilitate the tracking process, but only when the total number of objects (from a previously processed frame) varies from the total number of silhouettes in the current processed frame. Figure 1 is a flow diagram of the entire system, showing the split/merge components implemented on FPGA. A history of all objects in the camera view is maintained. For each frame the number of silhouettes extracted after the connected component analysis is compared with the number of known objects. If the number is unequal, the binary signatures (from the histogram data) of the silhouettes, as well as the objects are fed into the appropriate block on the FPGA for further analysis. The output of the FPGA describes when:

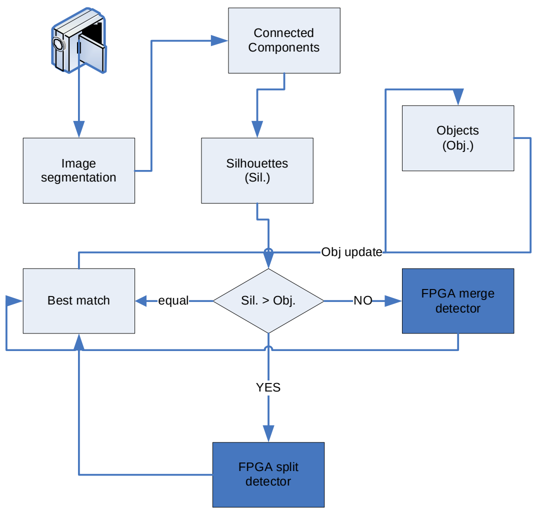

Figure 1. A flow diagram of the overall system, showing when FPGA is used to detect split/merge in object tracking.

1. Two objects have merged into a single silhouette,
2. A single object has split into two silhouette,
3. A new object has entered the scene,
4. An object has left the scene.

4. Split/Merge with Binary Histogram

The splitting and merging of the objects during tracking is always a challenging problem. The reason is that most object detection algorithms can only detect a blob (silhouette) as one object, and can only provide the feature for this object. When two blobs merge into one blob it is difficult to make a decision based on the features. This problem becomes more challenging, especially in embedded object tracking systems, as the algorithms should be able to run in real-time.

In our system we use colour histogram features. We define a binary signature from the colour histogram of each blob that is not only robust and consistent across frames, but also easy to store and implement in the FPGA hardware. In the following, we can prove that the Hamming distances between the binary signatures of merged blob and original blobs are significantly different, and can be used to detect splitting and merging of the blobs.

Lemma 1 *Assume* $x \in \{0, 1, ..., 2K - 1\}$ *and* $y \in \{0, 1, ..., 2K - 1\}$ *(*K *is a positive integer) are two i.i.d. random variables with discrete uniform distribution, then their sum* $z = x + y$ *has a discrete probability distribution with* $z \in \{0, 1, ..., 4K - 2\}$*. Define* $\tilde{x}$ *is a binary variable based on* x *and its mean* $\bar{x}$ *value,*

$$\tilde{x} = \begin{cases} 1 & x \geq \bar{x} \\ 0 & x < \bar{x} \end{cases} \tag{1}$$

Then the probability that $\tilde{z}$ *and* $\tilde{x}$ *are not equal is* $1/4$*.*

$$P\{\tilde{z} \neq \tilde{x}\} = 1/4 \tag{2}$$

Because the distribution of x, y are known, and $z = x+y$ is a simple variable, this lemma can be proved by counting the mismatch of the $\tilde{x}$ and $\tilde{z}$ where $\bar{z} = \bar{x} + \bar{y}$. Of course, the following is also true:

$$P\{\tilde{z} \neq \tilde{y}\} = 1/4 \tag{3}$$

If we have two i.i.d. sequences $X = \{x_1, x_2, ..., x_N\}$ and $Y = \{y_1, y_2, ..., y_N\}$ in which $x_i \in \{0, 1, ..., 2K - 1\}$ and $y_i \in \{0, 1, ..., 2K - 1\}$ have same discrete uniform distributions, the sum of the two sequences $Z = \{z_1, z_2, ..., z_N\}$ where $z_i = x_i + y_i$ will have the associated result, because every component has the result of the above lemma. As the Hamming distance of two binary sequences is the number of mismatches, we have the following theorem.

Theorem 1 *In the above condition, the mean values of the Hamming distance* Hmd *between the binary sequences* $\tilde{Z}$*,* $\tilde{X}$ *and* $\tilde{Y}$ *have the following results.*

$$\begin{aligned} mean(Hmd(\tilde{Z}, \tilde{X})) = N/4 \\ mean(Hmd(\tilde{Z}, \tilde{Y})) = N/4 \end{aligned} \tag{4}$$

In our tracking problem, we can treat two colour histograms of two original blobs as X and Y. Then the merging blob can be considered as $Z = X + Y$ at the frame where merging and splitting occurs. We can find that both the Hamming distance of the binary sequences $\tilde{Z}$, $\tilde{X}$ are very big. For example, when $N = 768$, the mean hamming distance will be 192. So we can define a threshold on the Hamming distance to detect splitting and merging.

For a sparsely populated scene, one can track the centres of the blobs, frame-by-frame. Thus, within a local area only one merging or splitting event will normally occur. In this case, our method as presented is very efficient and accurate in detecting splitting and merging.

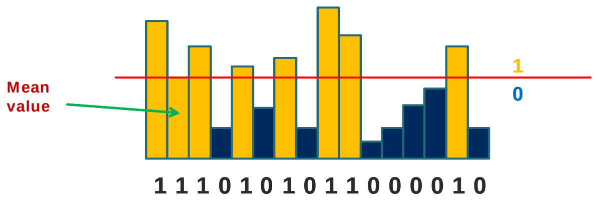

Figure 2. A sample 16 bin histogram, showing how the binary feature vector can be extracted. The bins in yellow (grey) are the bins greater or equal to the threshold value θ (mean of all bins) and the blue (black) bins are bins less than the threshold value. The yellow bins give a binary output of 1 and the blue a binary output of 0.

4.1. Feature Vector Construction

Using colour histograms to model scenes can be very expensive if the full 24-bit representation of the RGB pixels is used. Various encoding schemes can be used to reduce the number of bins required to represent the 24 bit colour information at the cost of reduced performance. The colour histogram itself can be used as a feature vector. When there are more bins in the histogram, the feature vector gives a more detailed descriptive representation with reduced processing time. Given the silhouette of a moving object after applying image segmentation (background differencing), a 768 bin histogram is generated; 256 bins for each of the RGB colour components. To convert this into a binary signature, the average θ for the 768 bins is computed as shown in equation 5 and any histogram bin with a value greater than or equal to θ is represented as binary 1 and binary 0 otherwise as shown in figure 2.

$$\theta = \frac{\sum_{i=1}^{768} bin_i}{768} \tag{5}$$

A binary feature vector (binary signature), $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ for $n = 768$ is generated as in equation 6.

$$x_i = \begin{cases} 1 & \text{if } bin_i >= \theta \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

4.2. Properties of Feature Vector

If the lightning conditions remain fairly constant, then the normalized colour histogram of a moving object also remains constant, irrespective of the position of the object to the camera. Similarly, the normalized histogram of a slow moving object appearing in the field of view of a camera will also remain constant. From equations 1-4, if two objects merge into a single silhouette and the overlapping region is fairly small as compared to the two constituent silhouettes, the histogram of the merged silhouette is approximately equal to the sum of the two individual silhouettes.

Also, if an object is fragmented into two parts, then the histogram of the joint silhouette is approximately equal to

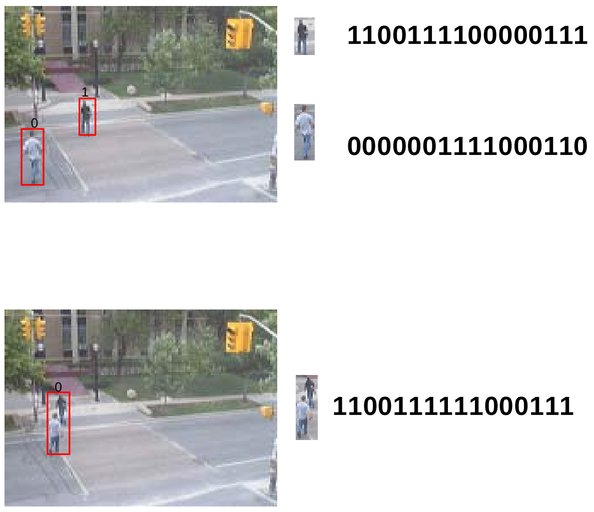

Figure 3. Two image sequences used to demonstrate the behaviour of binary vectors of tracked as object when they merge. The top image has two objects with different binary vectors, the bottom shows the two object merged with a binary vector similar to an $\oplus$ of the binary vectors of the two constituent objects.

the histogram of the sum of the fragmented parts. Binary histogram feature (binary vector) of a merged silhouette is approximately equal to the logical OR ($\oplus$) of the constituent silhouettes. Thus, an $\oplus$ of binary vectors of two fragments of a silhouette will be similar to the binary vector of the joint silhouette. Using binary signatures, a natural approach to measure similarity is the Hamming distance measure.

Given two sets of binary sequences $\mathbf{O_{f-1}} = (\mathbf{o_1}, \mathbf{o_2}, \ldots, \mathbf{o_n})$ and $\mathbf{S_f} = (\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_m})$, where $\mathbf{S_f}$ is the set of binary vectors collected from frame f, representing the m distinct silhouettes and $\mathbf{O_{f-1}}$ is the updated history of objects collected over $f-1$ frames for n objects. Each element of the set is a binary vector with 768 bits, representing the 768 colour histogram bins, i.e $\mathbf{o_1} = (o_1[0], o_1[1], \ldots, o_1[767])$. Figure 3 shows two image sequences with the corresponding binary vectors of the tracked objects. In the top frame, two objects have successfully been segmented as they are far apart. In the bottom frame, the two objects overlap and hence segmented as single object (a merge). The binary vector extracted from the bottom frame is approximately the $\oplus$ of the two binary vector from the top image.

When two objects enter the camera view next to each other (overlap) the tracking system will attempt to track them as they appear in the first frame. It is also possible for the object to move apart in the scene; creating an extra object for the tracker. The binary vector of the initially tracked object is expected to be approximately equal to the $\oplus$ of the two objects. Section 6, shows the result of this implementation on real image sequences. The best-matching process is significantly reduced if split/merge is detected in a pre-processing stage.

Size of binary vectors	768 bits
Maximum Silhouette in a frame	5
Maximum stored Objects	5
Threshold value	90

Table 1. Specification of the FPGA based split/merge implementation. The design has been tested with a maximum of four objects in the camera view.

5. FPGA Implementation

The most critical aspect of any hardware design is the selection and design of the architecture that provides the most efficient and effective implementation [5]. Base on binary vectors of moving objects, we implement an efficient FPGA based architecture for the fragmented objects, grouped objects, new objects and objects leaving the camera view. The specifications of the circuit implemented on FPGA is given in table 1 with its corresponding block diagram in figure 4. The circuitry is made up of five basic blocks: the comparison, logical OR, Hamming distance computation, Minimum Hamming distance computation and thresholding blocks. The thresholding block is triggered when the minimum Hamming distance is computed. Details of the five basic blocks are presented in the following sections.

5.1. The Comparison block

This is used to check the number of input objects and silhouettes to determine which block to trigger. A split might have occurred if the number of silhouettes exceeds the number of tracked objects. Similarly, a merge might have occurred if the number of objects exceeds the number of silhouettes. The design is limited to a maximum of four objects and five silhouettes for the split block, and a maximum of four silhouettes and five objects for the merge block. This can be scaled up to any number, depending on the available FPGA resource and most importantly if the scene is not crowded.

5.2. Logical OR block

This block is used to compute a logical OR between all pairs of silhouettes for the split detector (and all pairs of objects for the merge detector). For a maximum of five silhouettes, there are 40 possible pairs. The logical OR between all the 40 pairs are computed in parallel and takes exactly 768 clock cycles. The number of cycles can further be reduced if the storage structure is changed.

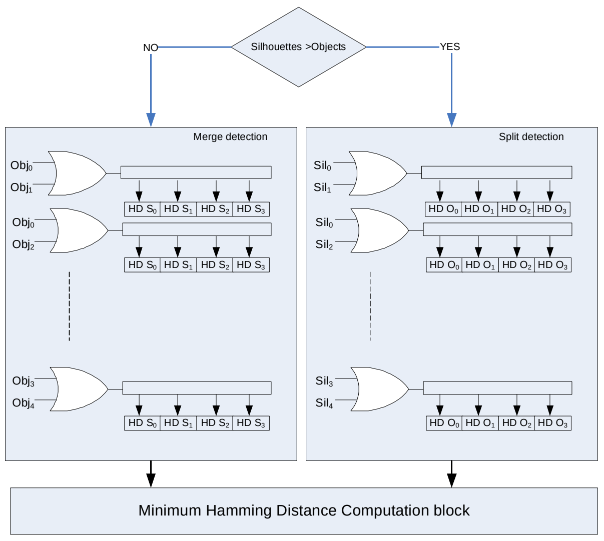

Figure 4. A block diagram of the FPGA design. Depending on the total number of silhouettes and objects, the split or merge block is used. For the split block, thus when the number of silhouettes is greater than the number of objects, a logical OR of all pairs of binary vectors id first computed. This is followed by the computation of the Hamming distance between every OR results and all binary vectors of the objects. The minimum Hamming distance is computed to establish the pair of split silhouettes. If the minimum Hamming distance is greater than the threshold value, a new objet is considered to have entered the scene. Similarly, the merge block is used to determine a possible merge or an object leaving the scene.

5.3. Hamming distance computation

This block is used to compute the Hamming distance between each logical OR output and all binary vectors of the objects (for the split detection) and all binary vectors of the silhouettes (for the merge detection). The Hamming distance between a logical OR output $\mathbf{x}_i$ and an object $\mathbf{o}_j$, as shown in equation 7 is a bitwise operation, and hence takes as many clock cycles as there are bits in the binary vector. Since the Hamming distance for all the 40 logical OR outputs are computed in parallel, it takes exactly 768 clock cycles to compute the Hamming distance for all the objects and silhouettes.

$$H_{ij} = \sum_{k=1}^{768} H_{ijk} \tag{7}$$

where k iterates through the bits in the binary vector and $i \in (1 \cdots 40)$ addresses the logical OR output.

5.4. Minimum Hamming distance

This block is used to identify the minimum value of all the Hamming distances computed in the previous section. The hardware design, as shown in figure 5, uses a series of comparators to select the minimum of a pair of two input Hamming distances. For an implementation with 40 values, the design takes exactly seven clock cycles to compute the minimum Hamming distance.

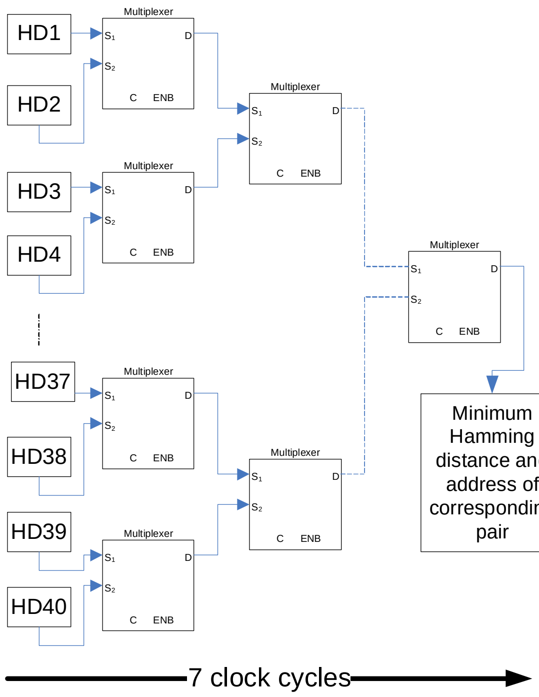

Figure 5. Structure of the minimum Hamming distance computation block. This block uses seven cycles to compute the minimum Hamming distance. The first cycle requires twenty comparators, which is halved every cycle to one in the seventh cycle.

5.5. Thresholding block

This block compares the output of the minimum Hamming distance block to make a decision. The output of this block is one of four values:

- 0 (split) – when the minimum Hamming distance from the split block is less than the threshold.
- 1 (new object) – when the minimum Hamming distance from the split block is greater than the threshold.
- 2 (merge) – when the minimum Hamming distance from the merge block is less than the threshold.
- 3 (object left) – when the minimum Hamming distance from the merge block is greater than the threshold.

When there is a split or merge, the split pair or merged silhouette is also reported.

The split/merge architecture discussed here has been implemented on a Xilinx Virtex-4 FPGA chip (XC4VLX160) with approximately 152,064 logic cells with embedded RAM totalling 5,184 Kbits. The design and verification was accomplished using the Handel-C high level descriptive language. Compilation and simulation were achieved using the Mentor Graphics DK design suite. Synthesis – the translation of abstract high-level code into a gate-level net-list – was accomplished using Xilinx ISE tools.

The entire design has been clocked at 40MHz, making it possible to process up to 25,000 binary vectors of size 768 bits in a second. The clock frequency of 40MHz also includes the design for controlling the external logic for the display and the input. This is the actual hardware test and the most stable clock frequency. The frequency could be much higher without the requirement to interface these devices. Table 2 gives the details of the resource utilization of the FPGA implementation.

Resource		**Total Used**	
Name	**Total**	**Used**	**Per.(%)**
Flip Flops	135,168	5,095	4
4 input LUTs	135,168	18,387	13
bonded IOBs	768	147	19
Occupied Slices	67,584	13,468	17
RAM16s	288	38	13

Table 2. Implementation results for the split/merge, using Virtex-4 *XC4VLX160*, package *FF1148* and speed grade *-10*.

6. Experimental Results

We evaluate the performance of the binary vector based split/merge algorithm with five different scenes, each collected over a period of 60 minutes. The total number of objects in any of the scenes is less than five. One of the five image sequences has been taken from an indoor environment with the other four from outdoor. The total number of frames used in our experiments is approximately $180,000$ with 78 objects, mostly persons. The tracking algorithm presented in [14] is used to extract binary histogram of moving blobs. Blobs with less than 768 pixels are removed as noise.

The main aim is to establish the characteristics of the binary histogram features of moving objects in the scene. We want to establish the similarity between the binary vectors for a single non-ridged object over a period of time, for its entire existence in the camera scene. Again, we will like to compute the joint binary vector of objects to estimate their similarity, should they merge. Figure 6, shows two images each from all the five test sequences. The binary vectors of objects are compared from frame-to-frame to determine the threshold value for the Hamming distance. Typical, the Hamming distance between the binary vectors of the same object across frames should be approximately equal to zero. Due, to the non-ridged nature of the objects being tracked, the colour histogram may vary from frame-to-frame and hence the binary vectors.

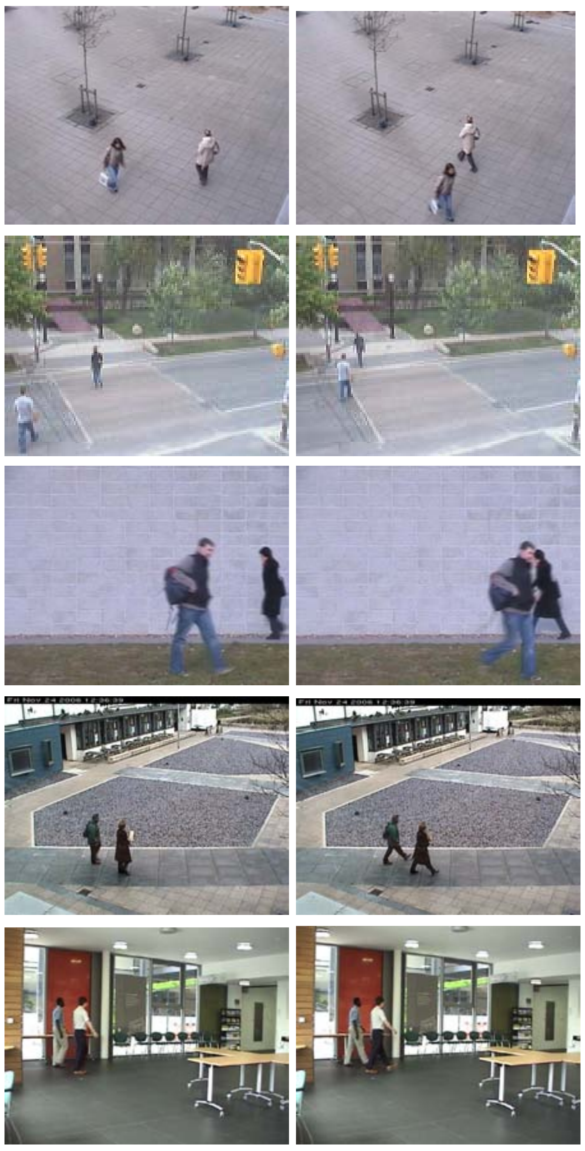

Figure 6. Two frames each taken from all the five scenes used in the experiments.

Table 3, shows the Hamming distance between the two objects from the top row of figure 6. The Hamming distance is computed for each tracked object between consecutive frames, from frame 25 to 29. The Hamming distance between the joint binary vector of the merged objects, in frame 30 and the binary vectors for the two individual objects taken from frame 29 is *56*. Similarity between objects and silhouettes is established with the minimum Harming distance. A threshold value of *90* has been chosen empirically after test with all the five image sequences. It has also

	Hamming Distance		
Frame #	Obj.0	Obj.1	Obj.0 vrs Obj.1
25	83	87	121
26	87	84	127
27	75	79	118
28	72	72	114
29	67	70	109

Table 3. Table showing the Hamming distance between two objects from frame-to-frame to establish when there is a split/merge, using the binary vectors generated from their colour histograms.

been established from the experiments that, Hamming distance between different objects exceed the threshold from frame-to-frame.

7. Conclusion

We have demonstrated a vision based framework for detecting grouped and object fragmentation using binary vectors generated from colour histograms. The use of binary vectors makes it possible to efficiently implement the design on FPGA to run in real-time. We have also demonstrated how the execution time for the best-match process used by most silhouette-base tracking algorithms can be reduced with the detection of split/merge objects prior to it. Work is ongoing on the design of an efficient similarity measure for use in the best-match process, to enable a complete tracking system fully implementable on FPGA.

Acknowledgements

This work was supported by TSB under the BRAINS Project. Special thanks goes to the project team; Prof. Derek Sheldon, Nigel Priestley, Mervyn Hobden, Peter Hobden and Cy Pettit.

References

[1] K. Appiah, A. Hunter, J. Owens, P. Aiken, and K. Lewis. Autonomous real-time surveillance system with distributed ip cameras. In *Third ACM/IEEE International Conference on Distributed Smart Cameras*. IEEE Computer Society, 2009.

[2] B. Bose, X. Wang, and E. Grimson. Multi-class object tracking algorithm that handles fragmentation and grouping. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2007.

[3] Y. Boykov and D. P. Huttenlocher. Adaptive bayesian recognition in tracking rigid objects. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:2697, 2000.

[4] T. J. Broida and R. Chellappa. Estimation of object motion parameters from noisy images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(1):90–99, 1986.

[5] C. Chang, M. Shibu, and R. Xiao. *Self Organizing Feature Map for Color Quantization on FPGA*. Springer, 2006.

[6] J. U. Cho, S. H. Jin, X. D. Pham, D. Kim, and J. W. Jeon. Fpga-based real-time visual tracking system using adaptive color histograms. *Proc. of the IEEE International Conference on Robotics and Biomimetics*, pages 172–177, 2007.

[7] R. T. Collins. Mean-shift blob tracking through scale space. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:234, 2003.

[8] M. Isard and A. Blake. *International Journal of Computer Vision*, 29:5–28, 1998.

[9] A. Jacquot, P. Sturm, and O. Ruch. Adaptive tracking of non-rigid objects based on color histograms and automatic parameter selection. In *Proceedings of the IEEE Workshop on Motion and Video Computing*, pages 103–109. IEEE Computer Society, 2005.

[10] S. Johnsen and A. Tews. Real-time object tracking and classification using a static camera. *Proceedings of IEEE International Conference on Robotics and Automation, workshop on People Detection and Tracking*.

[11] T. S. Ling, L. K. Meng, L. M. Kuan, Z. Kadim, and A. A. B. Al-Deen. Colour-based object tracking in surveillance application. *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol I*, 1, 2009.

[12] M. McErlean. An fpga implementation of hierarchical motion estimation for embedded object tracking. *International Symposium on Signal Processing and Information Technology*, 0:242–247, 2006.

[13] F. Muhlbauer and C. Bobda. Design and implementation of an object tracker on a reconfigurable system on chip. *IEEE International Workshop on Rapid System Prototyping*, 2006.

[14] J. Owens, A. Hunter, and E. Fletcher. A fast model-free morphology-based object tracking algorithm. In *Proceedings of the British Machine Vision Conference*. British Machine Vision Association, 2002.

[15] V. Takala and M. Pietikainen. Multi-object tracking using color, texture and motion. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[16] Q. Wang and Z. Gao. Study on a real-time image object tracking system. In *Proceedings of the 2008 International Symposium on Computer Science and Computational Technology*, pages 788–791. IEEE Computer Society, 2008.

[17] P. Withagen and F. G. K. Schutte. Likelihood-based object detection and object tracking using color histograms and em. *Proceedings of International Conference on Image Processing.*, 1:589–592, 2002.

[18] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[19] J. Xu, Y. Dou, J. Li, X. Zhou, and Q. Dou. Fpga accelerating algorithms of active shape model in people tracking applications. *Digital Systems Design, Euromicro Symposium on*, 0:432–435, 2007.

[20] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4):13, 2006.