Embedded Vision System for Atmospheric Turbulence Mitigation

Ajinkya Deshmukh[1], Gaurav Bhosale, Swarup Medasani[2], Karthik Reddy,
Hemanthakumar P, Chandrasekhar A, Kirankumar P, Vijayasagar K
Uurmi Systems Pvt. Ltd., Hyderabad, India
{[1]ajinkyad,[2]shanti}@uurmi.com

Abstract

Outdoor surveillance systems that involve farfield operations often encounter atmospheric turbulence perturbations due to a series of randomized reflections and refraction effecting incoming light rays. The resulting distortions make it hard to discriminate between true moving objects and turbulence induced motion. Current algorithms are not effective in detecting true moving objects in the scene and also rely on computationally complex warping methods. In this paper, we describe a real time embedded solution connected with traditional cameras to both rectify turbulence distortions and reliably detect and track true moving targets. Our comparisons with other methods shows better turbulence rectification with less false and miss detections. FPGA-DSP based embedded realization of our algorithm achieves nearly 15x speed-up along with lesser memory requirement over a quad core PC implementation. The proposed system is suitable for persistence surveillance systems and optical sight devices.

1. Introduction

Long-range surveillance systems with high zoom factors often suffer from the effects of atmospheric turbulence. Atmospheric turbulence is a spatio-temporal phenomenon induced from random fluctuations of the light ray alongs its path to the image sensor. This introduces geometric distortions as well as local scene blurriness. Standard single image de-blurring and de-convolution method can correct only blur but not the distorted geometry. Fundamental turbulence rectification approaches discussed in the literature are multi-frame image reconstruction methods [1–4]. These methods are sequential consisting of some combination of deformable registration, local image fusion and image de-blurring. Although these methods result in detail preservation and turbulence rectification they operate with high computational load, latency, and memory requirement and are mostly suited for static scenes with less focus on preserving the details of moving targets. The complex wavelet transform based registration and fusion approach in [4] requires many past and future frames for processing along with a need for manual selection of moving object regions. The method by Fishbain et al. [5] attempts to preserves real object motion while eliminating turbulence using complex registration, optical flow, and fuzzy logic rules to integrate moving objects. Though this method produces an output sequence that preserves moving objects, there is still residual turbulent motion left in the scene and the turbulence effects on objects are also not rectified. Another drawback of this method is that it presents a higher computational burden when the area occupied by the moving objects is high.

Recently, Oreifej et al. [6] proposed a variant of robust principal component analysis based method to recover the stable background and moving objects simultaneously. Optical flow based weighting cues are used to separate out true objects from turbulent motion. The authors report good results for a few datasets, where the levels of turbulence were decreased by multi-frame averaging. This pre-processing step clearly presents a problem when there are medium or fast moving objects in the scene. This method is computationally intensive, cannot handle high levels of turbulence, has high memory requirements and relies heavily on the output of optical flow methods, which are found to be unreliable for this particular application [15]. Another joint video sharpening and temporal diffusion in FFT domain was proposed to mitigate turbulence [7]. Here, individual frames were sharpened using Sobelov gradient filter and then these frames were used to remove temporal distortions using a Laplacian temporal diffusion filter. The authors show improved results at the cost of using a lot of frames thus introducing a latency into the system operation. Another system described in [8] use GPU-based software system for real time turbulence mitigation at 30-60Hz for HD quality. Although turbulence rectification is good the ability to handle moving targets is unknown. However most of real-time application demands stand-alone, low power systems for remote operation.

Tracking moving objects in less chaotic scenes has been widely investigated [9, 10]. Most of the methods however,

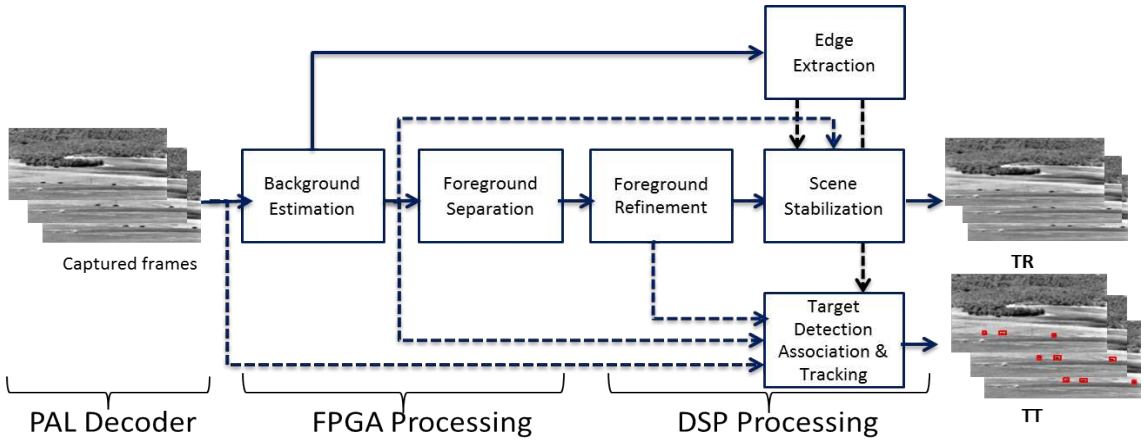

Figure 1: Functional block diagram of RESTORE system

do not consider the influence of turbulence induced distortions. The authors in [11] compared state-of-the-art tracking methods for a variety of scenarios. Among them, a method named SubSENSE [12] secured the top position for the turbulence category. This method improves upon Vibe [13] and PBAS [14] by including texture LBSP feature. Computation of these texture features is expensive making real time implementation challenging. In summary, under turbulent conditions, background subtraction methods alone cannot be relied upon to detect the moving targets reliably. Therefore there is a need a less complex system with minimum latency and memory requirement.

In this paper, we fill this gap by presenting an embedded approach for both turbulence rectification and moving target detection under varying degrees of turbulence. We abbreviate our system as 'RESTORE'(atmospheRic turbulEnce diStorTiOn REctification). Turbulence rectification is achieved by leveraging an important observation that most of turbulence distortions are visible at image structures. Target detection and tracking is done by analyzing blobs overlap with edges as well as statistical properties of blobs. Using a heterogeneous architecture with FPGA and a DSP where the FPGA is used for modeling the background and foreground and the DSP is used for edge detection, foreground pre-processing, and blob analysis. The entire system is controlled by serial processor (ARM Cortex A8). The block diagram of our hardware system is shown in Figure 1. This paper includes following contributions:

1. Real-time atmospheric turbulence mitigation with moving object preservation that exceeds the performance of methods in the literature
2. Modified detection and tracking of moving objects in turbulence for real-time application that has not been shown earlier to the same extent
3. FPGA-DSP based hybrid implementation to realize a complex algorithm in hardware
4. Hardware design, development with separate processing and IO functionalities for enabling a real-time embedded version of the system

The rest of the paper is organized as follows. Section 2 describes proposed system architecture with FPGA and DSP implementations along with serial controlling. In addition, heterogeneous acceleration platform selection and requirement is also discussed. In section 3, results analysis is given. Section 4 concludes our work with discussion about future enhancements.

2. Proposed System Architecture

Our system works in two independent modes and uses a composite FPGA-DSP architecture. First mode is Turbulence Rectification (TR) mode and second mode is Target Tracking (TT) mode. TR mode rectifies turbulence distortions without harming real moving objects whereas TT mode detects and tracks the true moving objects in turbulence. The system takes analog input (PAL resolution) from the camera and core algorithm processing takes place to give processed PAL output to display as shown in Figure 1. Both the TR and TT modes require foreground and background images for separating turbulence blobs from real moving blobs. So these common background estimation and foreground generation modules are implemented on FPGA closer to the decoded input video stream. Other modules such as edge detection, turbulence rectification and target tracking modules are developed on DSP to achieve better throughput and balance the computational load.

2.1. Heterogeneous DSP-FPGA Platform

We proposed to use the hybrid architecture (shown in Figure 2) to accommodate for memory, precise floating

point operations and achieve real-time performance. This base architecture includes ARM enabling of PAL decoder for capturing analog input, FPGA decoding of data, FPGA-DSP data transfer, video output driver for PAL and output routing to analog video display.

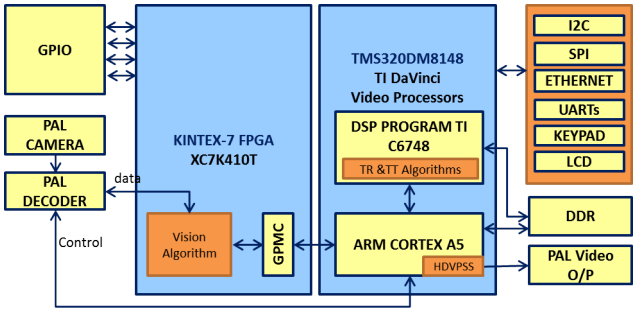

Figure 2: Heterogeneous DSP-FPGA architecture of RESTORE system

FPGA is used because of its inherent parallel architecture. In addition, because our system uses spatio-temporal analysis and needs more memory for storing temporal frames. To avoid the DDR accessing penalty we had to choose Kintex 7-XC7K410T because it has sufficient block RAMs for storing of five frames (I_t,I_{t-1},I_{t-2},B_t,$CbCr_t$) which are used for foreground refinement calculations.

For the DSP processor, we used the TMS320DM8148 DaVinci video processor which also has a co-located dual ARM cortex A8 processor. This ARM cortex A8 processor is used for streaming application and any other controlling software. We leveraged the fixed and floating point operations, single instruction multiple data architecture helps in pipelining and vectorization, and cacheable DDR memory access strengths of the DM8148 platform. We have also leveraged the onboard optimized image and video libraries such as VLIB (ex. Blob analysis), IMGLIB(ex. Median filter), MATHLIB (ex. angle calculations) as well as some c-callable intrinsic assembly functions for performing floating point operations.

Our processing flow diagram can be explained using Figure 3 which shows that many parallel step are incorporated to achieve better throughput. Initially after vertical blank (start of active video signal) we will get actual data stream which will be separated and stored into memory. By using ping pong operation that completes in $40ms$ for every frame, Y data will be stored in true data port memories TDP1 andTDP2 (as shown in Figure 4). Thereafter FPGA processing and transfer to DSP takes place as explained in section 2.2. The entire FPGA processing and transfer takes approximately $35ms$. The FPGA does the following tasks in parallel, namely sending CbCr data to DSP-ARM shared memory, new frame capturing, foreground and background estimation, foreground bits packing, and transferring foreground and background images to shared memory. Once the data is available median filtering and dilation processing foreground and canny edge detection on background is performed. Note that, edge detection is performed on every frame for turbulence rectification and one in five frames for target tracking. Then foreground image blobs are extracted and target detection and tracking is completed. The resulting output is copied into shared memory first and then into display buffer which takes around $1.5msec$. The complete DSP processing takes nearly $35msec$. Although both FPGA and DSP are taking less than $40msec$ time ARM has to wait till completion of $40msec$ interval (in parallel) to maintain overall synchronization of the system. General Purpose Memory Controller (GPMC) interface is used for data transfer from FPGA to ARM/DSP. This parallel interface is highly preferred for image processing applications as burst data (ex. complete image) can be transferred in quick time. The line by line or pixel-wise data transfer from FPGA to DSP-ARM shared memory is not feasible as delay in establishing data or control route using GPMC will be high enough than transferring of the pixel.

The hardware of our system is designed using stacked multi-layer PCB's. This stacking of board helps in reduction of overall size ($120mm \times 90mm$) of the system. The advantage of separating processing board from peripheral interface board is to support various sensor interfaces and input/output formats without interrupting core algorithm processing. Processing board includes FPGA, DSP and ARM functionalities along with various memories whereas peripheral interface board includes PAL capture, PAL output, Ethernet streaming to remote PC, LCD, keypad, power supply etc. (refer Figure 2). We now go into more details on the FPGA and DSP implementation details.

2.2. FPGA Implementation Details

A block diagram description of the FPGA implementation block is shown in Figure 4. FPGA logic captures streamed data from PAL decoder using various synchronization signals. This data is in $CbY0CrY1$ interlaced YUV422 format, so it is separated into Y and $CbCr$ as turbulence is intensity distortion and not color distortion [1]. This Y is used to estimate the moving average background. Moving average background estimation is used over temporal frame averaging or temporal median filtering [5] to avoid latency and extra memory requirement. Further, standard background subtraction techniques are also not used as they use complex techniques such as histograms of background and images, LBSP features, etc. [12–15] and need extra memory as well as logic for storing and updating background model. Our aim is to get an approximate version of the background that will coarsely model the scene [15]. For every input I_t(intensity image at time step t, Y), the background B_t is updated as follows:

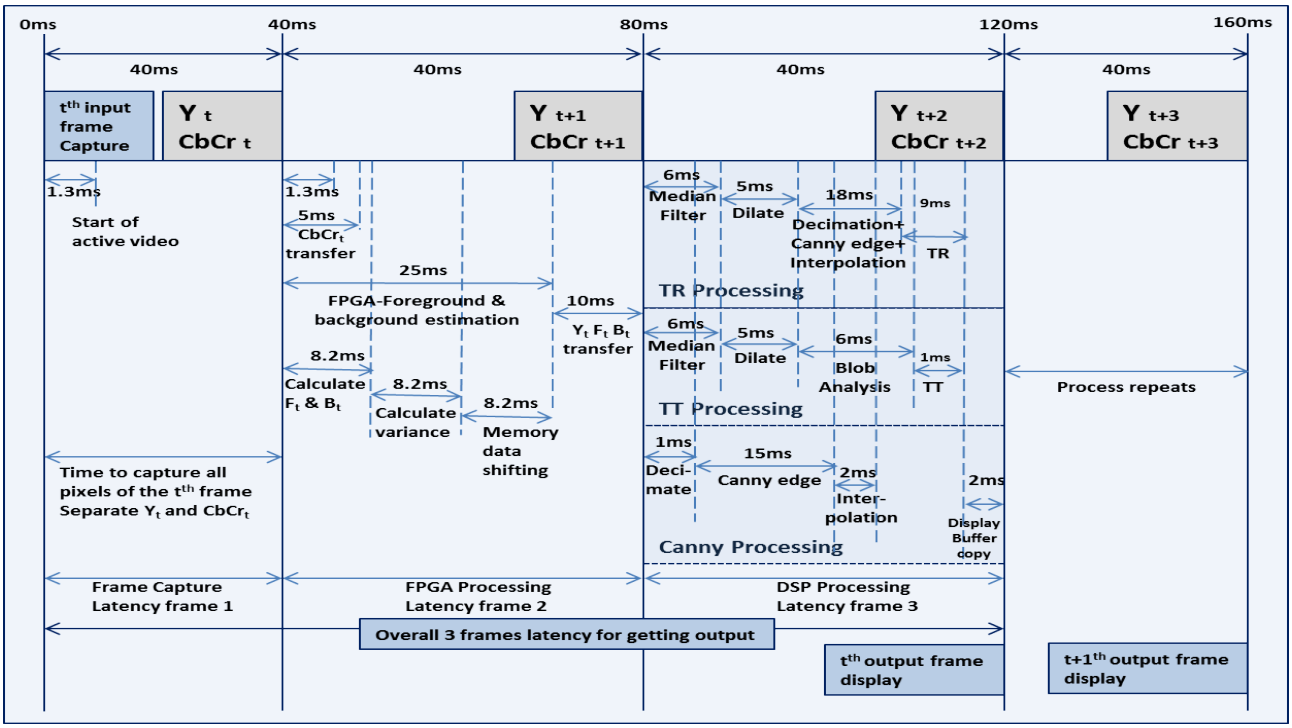

Figure 3: Pipelining approach for RESTORE system. Mentioned timings are for 50MHz FPGA Clock and 500MHz DSP Clock. Dashed vertical lines represent different time steps (not necessary to the scale).

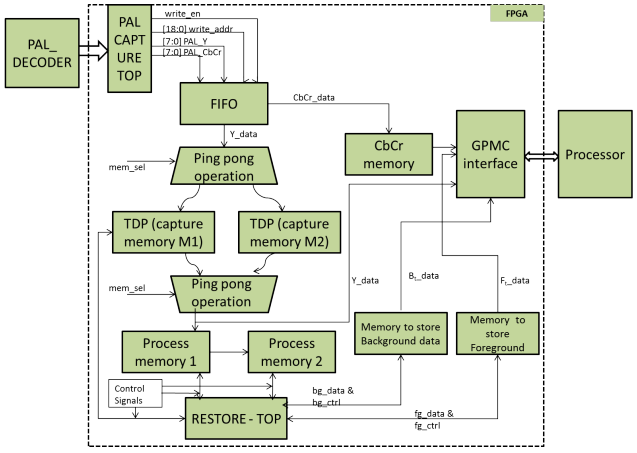

Figure 4: Background and foreground estimation-FPGA implementation

$$B_t = (1 - \frac{1}{f})B_{t-1} + \frac{I_t}{f} \tag{1}$$

Where, f is the frame number and acts as a weighting coefficient. This implies that background gets initialized as first input frame and slowly adapted to the current frame over time. After fixed set of frames (say 256) the weighting coefficient is fixed to avoid register bit width overflows. This also helps us in learning background quickly in the initial phase of the system. This calculated approximate background will be stored in FPGA on-chip memory for update. The important assumption made here is that for a static camera the background can change slowly over time. Sudden changes like environment lightning effects may not change background provided that it is not at an initial level. Further for cases where background objects start moving or foreground objects are merged into background, background may have some false signature of objects for few frames (if objects are moving with slow or medium velocity) that will eventually disappear after some frames. The common way of estimation of foreground is frame differencing. Some authors use concepts like signed differences to get the pattern of motion [16]. Similarly, we have used a two-level foreground exaction concept as shown below:

$$\begin{gathered} D_t^1 = |I_t - B_t| \\ D_t^2 = \\ |I_t - 0.5 * B_t - 0.25 * I_t - 0.125 * I_{t-1} - 0.125 * I_{t-2}| \\ D_t = \left|D_t^1 - D_t^2\right| \end{gathered} \tag{2}$$

Due to coarse background estimation, D_t^1 will help us to get difference image having true object blobs along with false turbulence blobs whereas D_t^2 leads to difference im-

age with mostly turbulence blobs (without real moving object blobs). All coefficients in the expression for D_t^2 are selected in such a way that it will lead to bit-wise shifting on FPGA. All the three input frames (I_t, I_{t-1}, I_{t-2}) are taken from processing memories. In addition, the shifting module is used for shifting the data between three processing memories for every start of frame SOF signal (refer Figure 4). The final difference image D_t leads to minimum number of false blobs. For initial two frames, input image is replicated twice to avoid initial latency of two frames.The background and difference calculation is done in parallel as previous stored background is used in difference image calculation module. Further, the threshold is calculated using the variance of this difference image and used to generate foreground image. We found that image variance is an effective measure to model the foreground intensity distribution. So outliers are rejected here and image binarization (F_t) is done as follows:

$$F_t = \begin{cases} 1, & D_t < T_t \\ 0, & Otherwise \end{cases} \tag{3}$$

Where, $T_t = 9 * v_d + \epsilon * D_t^{max}$ where v_d variance of D_t frame and $\epsilon * D_t^{max}$ is a small constant to suppress noise and compression artifacts. D_t^{max} is set to 255 to avoid maximum value calculations in FPGA. Here, for ease of FPGA implementation, T_t is approximated with its equivalent 8-bit fixed point value without compromising foreground result. Small foreground pixel errors will be eliminated in foreground post-processing step of DSP. Further, T_t is used from previous frame to avoid the delay of traversing all pixels to estimate the mean and then variance.

Then I_t, B_t, and F_t along with $CbCr_t$ memory are transferred to shared memory between ARM and DSP via GPMC interface. As F_t is binary mask it is packed as 64 bits and appended at the end of I_t to improve the transfer time. Then ARM based control logic transfers the shared memory address location details to DSP for further processing. Thereafter in DSP edge detection, turbulence rectification and turbulence tracking functionalities will be carried out.

2.3. DSP Implementation Details

In the DSP as a first step, based on provided shared memory address and fixed known image offsets, the data is retrieved. The advantage of moving average based background image is that there is absence of moving object. So if we find the edges of this image and compare with unpacked foreground image then we can separate out most of the false turbulence blobs based on the overlap criterion. We used canny edge detector over other edge operators due to its ability to suppress the non-edge noise. The standard canny edge method extracts edges E_t from background B_t as

$$E_t = canny(B_t) \tag{4}$$

In foreground processing, pixel level thresholding was used (in Section 2.2) so, foreground image may contain spurious turbulence pixel level distortions as well as there might be some holes in the estimated foreground blobs. So median filter and dilation is applied sequentially to suppress small blobs that are false detections and fill the missing areas of the true detected blobs. Once refined foreground (F_t), structured background (E_t) and frame history $(I_t, ... I_{t-k2})$ is available then atmospheric turbulence rectification (S_t) can be done using:

$$S_t = \begin{cases} (I_t + \cdots + I_{t-k_1+1})/k_1, & F_t = 1 \wedge E_t = 0 \\ (I_t + \cdots + I_{t-k_2+1})/k_2, & F_t = 1 \wedge E_t = 1 \\ B_t & \text{otherwise} \end{cases} \tag{5}$$

Note that here $k_2 > k_1$ for suppression of false turbulence blobs. The first criteria will reduce real moving object turbulence, whereas last will reduce overall non-object turbulence. Middle criterion preserves object boundaries overlapped with edges. This will also lead to smooth reconstruction at boundaries. We have found that $k_1 = 2; k_2 = 4$ for low and medium level of turbulence. For severe cases, $k_1 = 4; k_2 = 8$ is sufficient for stabilizing most of the dominant turbulence. All other previous frames are stored in available DDR memories with DSP and shifted accordingly with k_2^{th} frame discarded from memory. Spatial gradient filter is applied on generated S_t to remove blur caused due to temporal averaging. Here note that geometric distortions of turbulence effects will be eliminated by temporal filtering and local blur will be suppressed by gradient filtering [1]. This intensity output is combined back with the corresponding $CbCr_t$ memory to preserve color information. Then this output S_t(in $YCbCr$ format) is written into shared memory making it available for ARM to send to the display buffer. The process flow on the DSP side described thus far, is depicted in a block diagram in Figure 5.

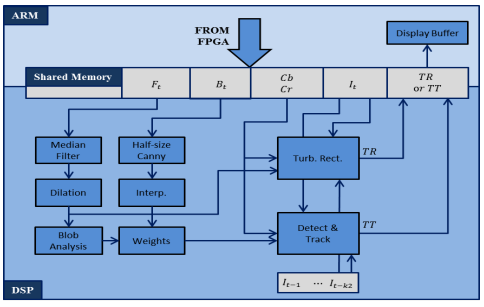

Figure 5: DSP-ARM shared memory architecture for faster communication from FPGA to DSP and DSP to display

For reducing canny edge detection computations, we have used background frame as half-sized image for canny edge detection processing. Advantage of using half-sized canny edge processing on background image is that dilation step can be avoided when edge image is interpolated to full size. The edges should be broad to account for small edge variations due to turbulence. This will reduce computational requirement of our system by atleast two folds. Further, this half-sized canny edge detection will operate once in the five frames as there is not much change in the background per frame. In addition, we will not process tracking algorithm for canny edge processed frame, instead, we output the previous track. DSP support for intrinsic image/video libraries (discussed in Section 2.1) also helped us in reducing the processing times for many of the functions as shown in Figure 6. Further, this graph indicates that we are able to achieve 5 to 15 times speed-up over C and DSP non-optimized versions for different modules.

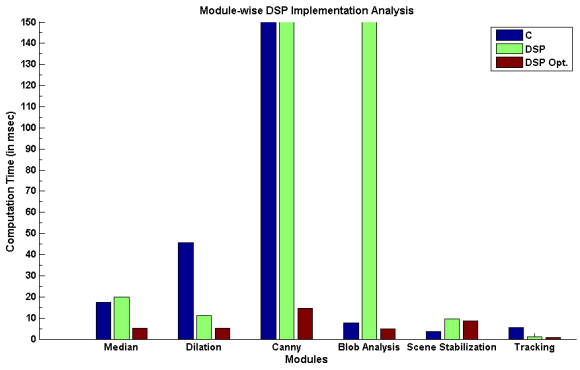

Figure 6: Module-wise DSP optimization analysis for computationally intensive functions. Graph is truncated to 150msec for better visualization

In target detection and tracking mode, blob features such as area, centroid, bounding box and label matrix are extracted from post-processed foreground image. The maximum number of blobs is set to 20 as this will help keep computations to a minimum as our tracking algorithm is dependent on the blob analysis step. Median filtering step of foreground pre-processing helps eliminate small blobs present in the foreground. This is followed by a statistical analysis of size, intensity, displacement vectors, and distance measures across frames to give weights to the blobs. These weights are used to find similarity between blobs at different time steps allowing for them to be reliably tracked. The cumulative effect of these weights leads to term similarity weight W_{sim}. As demonstrated in [15], turbulence is most predominant near edges; many of the false blobs are located around the edges in the image. So we have also used edge overlap weight W_{eo} that penalizes blobs that are located near the true edges in the image. Finally, a weight is assigned to the map from blobs in the current frame (the most recent frame which has been completely processed) to their potential correspondences in the next frame (the most recent frame that needs to be processed) is given by

$$W(p_k(i) \rightarrow N^j(p_k(i))) = exp\left(\frac{-d^j_{next}}{\sqrt{M^2+N^2}}\right) * [(1-\alpha)W_{eo}(p_k(i)) + \alpha W_{sim}(N^j(p_k(i)))] \quad (6)$$

Where $p_k(i)$ is i^{th} blob in current frame of size $M \times N$, $N^j(p_k(i))$ is j^{th} nearest neighbor from next frame, $j \in [1,5]$. α is a predefined constant learning rate to balance weights between edge overlapped real objects and false turbulence blobs. d^j_{next} is Euclidean distance between current blob and its jth neighbor in next frame. An exponential factor is added to further penalize distant blobs. Finally, the likelihood of this weight over few frames is observed before a detection decision is made. For detailed description of the equation (6), refer to our previous work[15]. Tracking algorithm used for DSP implementation is a modified version of method explained in [15]. A functional description of the weight estimation process is described in Figure 7. Although sometimes true object blobs may get penalized by edge overlap weight but similarity weight will differentiate it from false blobs. This shows that we have incorporated multiple criteria for validating real object motion.

The software libraries (intrinsic functionality) provided by Texas Instruments are used for foreground post-processing including median filter, dilation and blob analysis. Further, dilation and blob analysis is processed using packed 32-bit binary data for better optimization. In calculation of blob weights, the refined foreground is processed at locations where blobs are present. This minimizes the processing load and improves overall speed. Restriction on the processing of number blobs maintains processing time below the required frame rate. ARM based logic is then used for direct transfer of either turbulence rectification result or tracking results to display buffer.

3. Results and Discussion

The performance of our implementation for PAL resolution (720×576) with FPGA operating at 50MHz clock, DSP at 500MHz clock results in 25FPS frame rate for low to severe turbulence while tracking up to 20 moving objects. The overall latency of our system is three frames making it suitable for real-time surveillance systems. In our PC based implementation of algorithm with configuration of Intel core $i3$, 3.4 GHz, 8 GB memory the turbulence rectification system runs at 2FPS, and target detection while the tracking system runs at 1.5FPS. Comparatively, our dedicated embedded system achieves up to $15x$ speed-up. The resource utilization for FPGA is given in Table 1 and shows

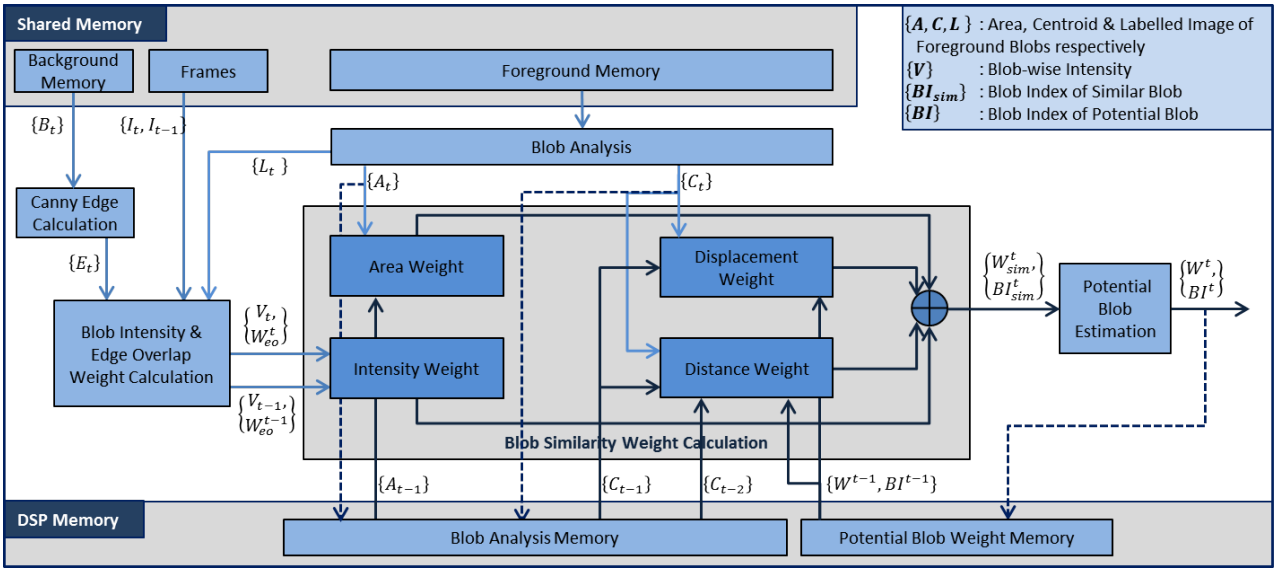

Figure 7: Function diagram of blob weight analysis in DSP

that sufficient block rams have been exploited to avoid DDR access penalties.

Table 1: Background estimation and foreground estimation FPGA resource utilization

Kintex 7 XC7K410T FPGA	Occupied	Available	Percentage Utilization
BIockRAMS	715	795	89.94 %
LUTS	8842	254200	3.48 %
DSP Slices	4	1540	0.26 %
Flip-flops	1783	508400	0.35 %

In addition, DSP/ARM utilization (Table 2) shows efficient usage of memories without overburdening the system.

Table 2: DSP resource utilization for RESTORE system

TMS320DM8148	Occupied (MB)	Available (MB)	Percentage Utilization
DSP Data Memory	56.41	80	70.00 %
DSP Program Memory	0.174		
ARM Data Memory	0.610	80	0.87 %
ARM Program Memory	0.085		
Shared Memory	0.881	16	5.5 %

We have used the following edge based no-reference turbulence measurement (EBTM) metric for performance evaluation of our algorithm. This metric calculates amount of turbulence present in the video based on root mean square error (RMSE) between two adjacent frames on the dilated edges without using any reference image.

$$EBTM(V_{in}) = \sum_{k=2}^{K} \frac{\| fr(k) - fr(k-1) \|_2}{\| 0.5 * (fr(k) + fr(k-1)) \|_2} \quad (7)$$

Here V_{in} is the input video under test and K is the total number of frames of the V_{in}. $fr(k)$ is dilated edge binary image of k^{th} frame. Further, 0.5 constant indicates that the final quantity is normalized to yield a consistent measure for all images. Although this metric is sensitive for large sized moving objects it works well for a variety of natural scenes and has good coherence to human subjective scores. Analytical results given in Table 3 shows that proposed system stabilizes turbulence from low to severe levels of turbulence (low = $EBTM < 0.25$; severe = $EBTM > 0.50$). Here, EBTM (V_{in}) indicates the amount of turbulence present in input video while EBTM (V_{out}) shows the amount of turbulence present in output video obtained from proposed rectification method. Qualitative results are also given in Figure 8. The comparison with [7] in Table 3 shows better turbulence rectification without harming real objects. Further, results for *turbulence3* dataset are biased by large fast mov-

Table 3: RESTORE system TR Mode performance evaluation and comparison with method [7]

Datasets	EBTM (V_{in})	EBTM (V_{out})	Percentage Rectification **RESTORE**	Percentage Rectification [7]
turbulence0	0.55	0.13	76%	75%
turbulence1	0.63	0.12	81%	77%
turbulence2	0.39	0.11	71%	78%
turbulence3	0.39	0.11	72%	82%

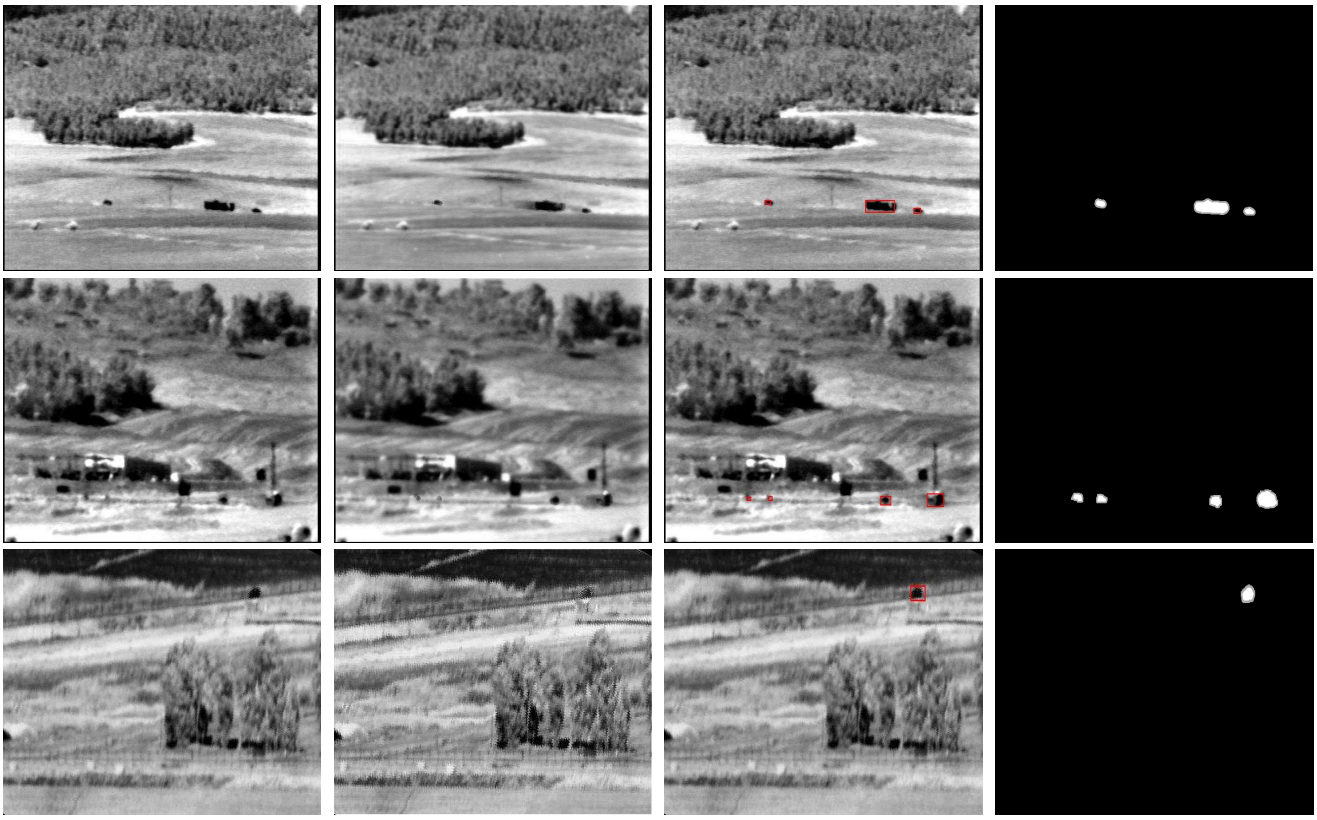

Figure 8: Qualitative analysis of RESTORE system algorithms.

ing objects.

Further, the potential blobs that survived after weight and temporal consistency analysis are binarized and compared with ground truth measurement given in [11]. We have compared all the datasets of turbulence category as given in Table 4 with [12] and found that our results are comparable in detecting real moving objects with ease of porting onto embedded platform.

Table 4: RESTORE system TT Mode performance evaluation and comparison with method [12]. All values are multiplied with factor 1000.

Measures	Datasets ($\times 10^{-3}$)					
	turbulence0	turbulence1	turbulence2	turbulence3	Avg. **RESTORE**	Avg. [12]
Recall	579.0	366.0	840.2	385.2	542.6	805.0
Specficity	999.9	999.8	1000.0	999.4	999.77	999.4
FPR	0.07	0.18	0.02	0.63	0.225	0.6
FNR	421.0	634.0	159.8	614.8	457.4	195.0
PWC	87.1	259.5	8.3	1052.0	351.72	152.7
F-1	714.6	517.4	881.4	541.0	656.85	779.2

For an ideal tracking system, Recall, Specificity and F-Measure should be high whereas FPR, FNR, and PWC should be low [11]. Further comparison of tracking algorithm performance with recent methods is given in our previous work [15]. As our algorithms are independent of sensor type (visible or IR) and resolution, we assume similar performance for digital camera (with and without compression). For comparison video results refer supplementary material provided with the paper.

4. Conclusion and Future Work

An efficient heterogeneous platform based real-time implementation of our RESTORE system mitigates the atmospheric turbulence distortions while reliably detecting and tracking true moving objects. We have exploited the prevalence of turbulence distortions near scene structures for both turbulence mitigation as well as moving target detection. Algorithmic comparison of our approach with existing literature has shown improved performance in accuracy, minimum latency, and fewer memory requirements.

We are working towards development of similar system on reduced form factor hardware so that it can be directly embedded into smart surveillance cameras and optical sight devices.

References

[1] X. Zhu and P. Milanfar. Removing atmospheric turbulence via space-invariant deconvolution. IEEE Trans.Pattern Anal. Mach. Intell., 35(157-170):14, 2013 1, 3, 5

[2] D. Gong, Y. Zhang ; S. Dang ; J. Sun. Neighbor combination for atmospheric turbulence image reconstruction. In The IEEE International Conference on Image Processing (ICIP), 2013 1

[3] C. S. Huebner and C. Scheifling. Software-based mitigation of image degradation due to atmospheric turbulence. Proc. SPIE, 7828:78280N-78280N-12, 2010 1

[4] N.Anantrasirichai, A.Achim, N. Kingsbury, D.Bull. Atmospheric turbulence mitigation using complex wavelet-based fusion. Image Processing, IEEE Transactions on, 2013 1

[5] B. Fishbain, L. P. Yaroslavsky, and I. A. Ideses. Real time stabilization of long range observation system turbulent video. J. Real-Time Image Processing, 2(1):11-22, 2007 1, 3

[6] O. Oreifej, X. Li, and M. Shah. Simultaneous video stabilization and moving object detection in turbulence.IEEE Trans. Pattern Anal. Mach. Intell., 35(2), 2013 1

[7] Y. Lou, S. Kang, S. Soatto and A. Bertozzi. Video Stabilization of Atmospheric Turbulence Distortion. Inverse Problems in Imaging, Special Issue in honor of Tony Chan, 7(3), pp. 839 - 861, August 2013. 1, 7

[8] *Atmospheric Turbulence Mitigation System.* [online], Available: http://www.atcomimaging.com/turbulence-mitigation (Date last accessed on Mar. 18, 2016) 1

[9] G. Baldini, P. Campadelli, D. Cozzi, and R. Lanzarotti.A simple and robust method for moving target tracking. In Proc. of the IASTED International Conference on Signal Processing, Pattern Recognition and Applications (SPPRA), pages 108-112, 2012. 1

[10] B. Benfold and I. Reid. Stable multi-target tracking in real-time surveillance video. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3457-3464, 2011. 1

[11] N. Goyette, P. Jodoin, F.Porikli, J. Konrad, and P. Ishwar. changedetection.net: A new change detection benchmark dataset. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pages 1-8, 2012. 2, 7, 8

[12] P. St-Charles, G. Bilodeau, and R. Bergevin. Flexible background subtraction with self-balanced local sensitivity. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2014. 2, 3, 8

[13] O. Barnich and M. V. Droogenbroeck. Vibe: a universal background subtraction algorithm for video sequences.IEEE Transactions on Image Processing, 20(6):1709-1724, 2011. 2, 3

[14] M. Hofmann, P. Tiefenbacher, and G. Rigoll. Background segmentation with feedback: The pixel-based adaptive segmenter. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pages 38-43, 2012. 2, 3

[15] A. Apuroop, A. Deshmukh, and S. Medasani, Robust Tracking of Objects through Turbulence, In the Proceedings of the ACM 2014 Indian Conference on Computer Vision Graphics and Image Processing,pages 29,2014 1, 3, 6, 8

[16] Y. Yoo, T. Park. A Moving Object Detection Algorithm for Smart Cameras. In the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '08), pages 1-8, 2008 4