Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2016, Article ID 8457908, 16 pages
http://dx.doi.org/10.1155/2016/8457908

Research Article

FPGA-Based Real-Time Moving Target Detection System for Unmanned Aerial Vehicle Application

Jia Wei Tang, Nasir Shaikh-Husin, Usman Ullah Sheikh, and M. N. Marsono

Faculty of Electrical Engineering, Universiti Teknologi Malaysia (UTM), 81310 Skudai, Johor Bahru, Malaysia

Correspondence should be addressed to Nasir Shaikh-Husin; nasirsh@fke.utm.my

Received 15 November 2015; Revised 5 February 2016; Accepted 10 March 2016

Academic Editor: João Cardoso

Moving target detection is the most common task for Unmanned Aerial Vehicle (UAV) to find and track object of interest from a bird's eye view in mobile aerial surveillance for civilian applications such as search and rescue operation. The complex detection algorithm can be implemented in a real-time embedded system using Field Programmable Gate Array (FPGA). This paper presents the development of real-time moving target detection System-on-Chip (SoC) using FPGA for deployment on a UAV. The detection algorithm utilizes area-based image registration technique which includes motion estimation and object segmentation processes. The moving target detection system has been prototyped on a low-cost Terasic DE2-115 board mounted with TRDB-D5M camera. The system consists of Nios II processor and stream-oriented dedicated hardware accelerators running at 100 MHz clock rate, achieving 30-frame per second processing speed for 640 × 480 pixels' resolution greyscale videos.

1. Introduction

Unmanned Aerial Vehicle (UAV) plays an important role in mobile aerial monitoring operations and has been widely applied in diverse applications such as aerial surveillance, border patrol, resource exploration, and combat and military applications. Due to its mobility, UAV has also been deployed for search and rescue operation [1] by acquiring high-resolution images in disaster area. Apart from that, several researches [2, 3] have also been done on traffic monitoring using UAV. As most monitoring systems require detection and tracking object of interest, moving target detection is a typical process in UAV monitoring system [4].

Moving target detection is the process of locating moving objects (foreground) residing in the static scene (background) from a series of visual images captured from a camera. As displacement of object in subsequent video frames defines its movement, at least two successive video frames are required for processing. An object is defined as a moving target if it is located in two different positions corresponding to the background from two selected frames taken at different time intervals. Thus, a background model is required to represent the static scene from incoming video frames prior to the segmentation of moving object.

Background model can be categorized based on the type of camera movement [5], including stationary camera, pan-tilt-zoom camera, free camera motion with planar scene, and free camera motion with complex scene geometry. Detection and segmentation of moving objects in stationary background (static camera) can be performed easily using background subtraction technique [6–11], while image registration technique is required in moving background (moving camera) involving ego-motion (camera motion) estimation and compensation to align the backgrounds of selected video frames prior to object segmentation. The scene in aerial imagery in UAV video is assumed to be planar [12]. The ego-motion estimation for planar scene can be estimated using homography transformation such as affine model. Hence, moving object can be detected by registering the video frame to the estimated model and employing the background subtraction with this registered model. This approach does not consider the scene with significant depth variations as it causes incorrect registrations due to parallax.

Due to the complexity of computer vision algorithm, moving target detection in aerial imagery is a time consuming process. It is also not practical to rely on a ground processing station via radio link as video quality will greatly depend on the wireless communication speed and stability. In addition,

full autonomous UAV is desirable as it can operate and react towards detected target with minimal human intervention [13]. Thus, an autonomous UAV demands a system with high mobility and high computing capability to perform detection on the platform itself. The use of Field Programmable Gate Array (FPGA) will satisfy the low power consumption, high computing power, and small circuitry requirements of a UAV system. FPGA-based system is a good solution in real-time computer vision problem for mobile platform [14] and can be reconfigured to handle different tasks according to desired applications.

This paper presents a FPGA implementation of real-time moving target detection system for UAV applications. The detection algorithm utilizes image registration technique which first estimates the ego-motion from two subsequent frames using block matching (area-based matching) and Random Sample Consensus (RANSAC) algorithm. After compensating the ego-motion, frame differencing, median filtering, and morphological process are utilized to segment the moving object. The contributions of this paper are as follows:

(i) Development of real-time moving target detection in a System-on-Chip (SoC), attaining 30 frames per second (fps) processing rate for 640 × 480 pixels' video.

(ii) Prototyping of the proposed system in a low-cost FPGA board (Terasic DE2-115) mounted with a 5 megapixels' camera (TRDB-D5M), occupying only 13% of total combinational function and 13% of total memory bits.

(iii) Partitioning and pipeline scheduling of the detection algorithm in a hardware/software (HW/SW) codesign for maximum processing throughput.

(iv) Stream-oriented hardware accelerators including block matching and object segmentation module which are able to operate in one cycle per pixel.

(v) Analyzing detection performance with different density of area-based ego-motion estimation and frame differencing threshold.

The rest of the paper is as follows. Section 2 discusses the literatures in moving target detection. Section 3 discusses the moving target detection algorithm while Section 4 describes the SoC development and the specialized hardware architecture of moving target detection. Section 5 presents the detection result from the complete prototype. Section 6 concludes this paper.

2. Related Work

Moving target detection targeting for aerial videos or UAV applications has been widely researched in the past few decades. A framework consisting of ego-motion compensation, motion detection, and object tracking was developed in [15]. The authors used combination of feature and gradient-based techniques to compensate ego-motion while utilizing accumulative frame differencing and background subtraction to detect moving vehicle in aerial videos. The research in [16] presented two different approaches to detect and track moving vehicle and person using a Hierarchy of Gradient (HoG) based classifier. The work in [17] has proposed a moving target detection method that performs motion compensation, motion detection, and tracking in parallel by including data capture and collaboration control modules. Multiple target detection algorithm was proposed in [18], catering for large number of moving targets in wide area surveillance application. Moving target detection and tracking for different altitude were presented and demonstrated on UAV-captured videos in [19]. Feature-based image registration technique was proposed in [20] to detect moving object in UAV video. The authors utilized corner points in subsequent video frames as features to perform ego-motion estimation and compensation. In [21], a multimodel estimation for aerial video was proposed to detect moving objects in complex background that is able to remove buildings, trees, and other false alarms in detection. As these literature works focused on improving the detection algorithm for different cases and did not consider autonomous UAV deployment, they developed their system in a common desktop computer [17, 19–21] or Graphic Processing Unit (GPU) accelerated [22] environment.

Table 1: Comparison between related works on FPGA-based object detection system for different applications with proposed system.

Related work	Camera platform	Detection technique and application
[6–11]	Static camera	(i) Background subtraction (ii) Using GMM, ViBE, and so forth
[23]	Moving robot	(i) Detecting moving object using (ii) Optical flow and frame differencing
[24]	UAV	(i) Detecting and tracking object feature
[13]	UAV	(i) Car detection (ii) Based on shape, size, and colour
[25]	UAV	(i) Detecting moving object (ii) Using regional phase correlation (iii) Does not prototype the complete system
[26]	UAV	(i) Real-time ego-motion estimation
Proposed	UAV	(i) Moving target detection (ii) Using area-based image registration (iii) Prototyping the complete system

In the context of FPGA-based object detection system, most works in the literature were targeted for static camera [6–11] as illustrated in Table 1. They utilized background subtraction techniques such as Gaussian Mixture Model (GMM) and ViBE (Visual Background Extractor) to perform foreground object segmentation in static background video. The work in [23] has proposed FPGA-based moving object detection for a walking robot. They implemented ego-motion estimation using optical flow technique and frame differencing in hardware/software codesign system.

There are also several literatures proposing FPGA-based detection for UAV applications. The research in [24] has proposed a hardware/software codesign using FPGA for feature detection and tracking in UAV applications. The authors

implemented Harris feature detector in dedicated hardware to extract object features from aerial video while tracking of object based on the features is executed in software. Implementation of real-time object detection for UAV is described in [13] to detect cars based on their shape, size, and colour. However, both works in [13, 24] performed detection and tracking based on object features and did not focus on moving targets. A suitable moving target detection algorithm for FPGA targeting sense and avoid system in UAV has been proposed in [25] by using regional phase correlation technique but the authors did not prototype the complete system in FPGA device. In addition, research in [26] also presented the hardware design and architecture of real-time ego-motion estimation for UAV video. Hence, there are limited numbers of works in the literature focusing on the development of a complete prototype to perform real-time moving target detection for UAV applications using FPGA.

3. Moving Target Detection Algorithm

As UAV is a moving platform, the proposed moving target detection algorithm employs image registration technique to compensate the ego-motion prior to object segmentation. Image registration algorithms can be classified into feature-based and area-based (intensity-based) methods [27, 28].

In feature-based method, detected features such as corners [29, 30] or SURF [31] from two subsequent frames are cross-correlated to find the motion of each feature from one frame to another. Feature-based image registration is reported to have faster computation in software implementation as it uses only a small number of points for feature matching regardless of the number of pixels. The number of detected features is unpredictable as it depends on the captured scene of the frames, thus having unpredictable amount of computation and memory resource, making it difficult to be implemented in highly parallel hardware. Number of features can be reduced to a predictable constant with an additional step of selecting strongest features based on their score (i.e., feature strength) by sorting or priority queuing [24]. However, it presents some limitations as only pixels of the highly textured areas would be selected while neglecting the homogeneous area [32]. Moreover, feature-based method requires irregular access of memory which is not suitable for streaming hardware.

On the contrary, area-based technique construct point-to-point correspondence between frames by finding the most similar texture of a block (area) from one frame to another. It is suitable for parallelism and stream processing as it offers several benefits for hardware implementation:

(i) It has highly parallel operations that make it suitable for parallel processing in hardware implementation.

(ii) It allows simple control-flow and does not require irregular accessing of image pixels.

(iii) It has predictable memory requirement with fixed size of computation data.

The overall flow of the proposed algorithm is illustrated in Figure 1. It consists of two main processes, which are motion estimation and object segmentation. Area-based image registration is utilized in this work. The inputs to the system are two consecutive greyscale video frames, which are the current and the previous frames. First, block matching is performed on these two frames to produce point-to-point motion between frames. As aerial imagery in UAV video is assumed to have free camera motion with planar scene [5], affine model is employed to estimate the ego-motion. RANSAC is then used to remove insignificant motion (outliers) among all points, resulting in the ego-motion in terms of affine transformation matrix.

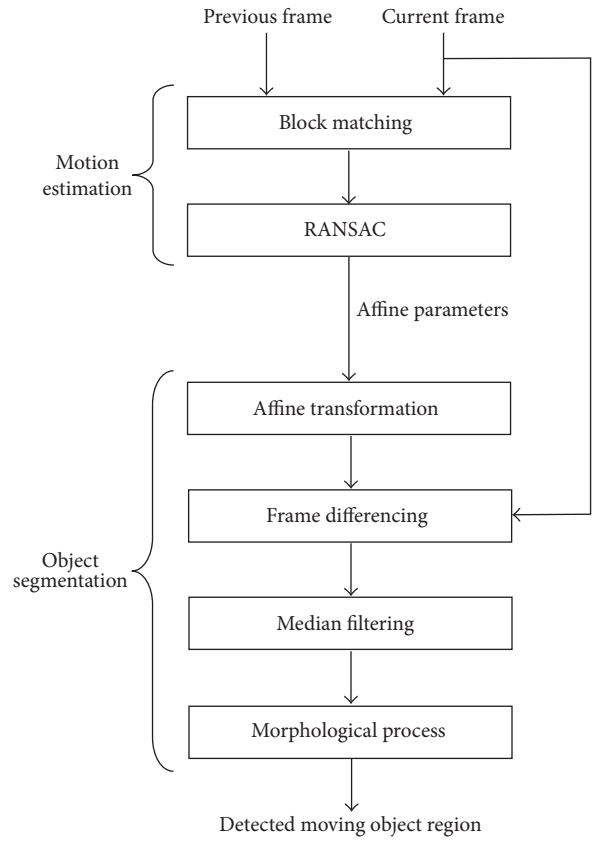


Figure 1: Overall algorithm of moving target detection using image registration technique.

After the previous frame is aligned with current frame using parameters in the affine transformation matrix, frame differencing can be performed with pixel-by-pixel subtraction on both aligned frames, followed by thresholding to produce a binary image. Median filtering and morphological processes are done on the binary image to remove noises, resulting in only the detected moving target.

The proposed algorithm is intended for SoC implementation consisting of a Nios II embedded software processor running at 100 MHz. However, most processes running on Nios II are slow and insufficient to achieve real-time capability. In order to realize a real-time moving target detection system, all processes in this work are implemented in fully dedicated

hardware accelerators except RANSAC, which is partially accelerated in hardware.

3.1. Block Matching. Block matching involves two steps: extraction and matching, where two consecutive frames are required. Extraction process will store several blocks or patches of image pixels from one frame as template, while matching process will find their most similar blocks in the second frame. By considering the center points of blocks as reference, this algorithm will yield numerous pairs of corresponding points which indicate the point-to-point motion (movement of the pixels) between two consecutive frames. The paired points from these two frames will be used in RANSAC to estimate the ego-motion.

Block extraction is the process of storing numerous blocks of 9×9 pixels from a predefined location from a video frame. These blocks will be used as templates in the matching process. The positions of the template blocks are distributed evenly over the image. There is no mathematical computation in the extraction process as it involves only direct copying of image patches from video stream into temporary memory.

Matching process plays the role of finding the most similar blocks from current frame for every extracted template block from the previous frame. This is done by correlating the template blocks with next frame to find their corresponding position based on similarity measure. Due to simplicity of hardware implementation, Sum of Absolute Difference (SAD) is chosen as the matching criterion for the correlation process. SAD will generate a similarity error rating of pixel-to-pixel correlation between each template block (from previous frame) and matching block (from current frame). SAD will yield zero result if both blocks are pixel-by-pixel identical.

Block matching is computation intensive as each template block has to search for its most similar pair by performing SAD with each block within its search region. Several search techniques had been proposed in the literatures to reduce the computation by minimizing the search region such as Three-Step Search Technique [33, 34], Four-Step Search Technique [35], and Diamond Search [36]. However, most of these techniques are targeted for general purpose processor which reads image in irregular way and are not suitable for streaming hardware architecture. This work uses traditional full search technique [37] as it is efficient to be performed in stream-oriented hardware due to its regular accessing of image.

The number of required matching computations is proportional to the number of blocks (density) and their corresponding search areas. Higher density of block matching provides more points for ego-motion estimation to reduce image registration error but with higher hardware cost requirement (number of hardware computation units). To reduce hardware cost, this work employs only a low density block (area-based) matching and does not estimate frame-to-frame motion of every pixel.

To further optimize hardware resources in stream-oriented architecture, best-fit and nonoverlapping search areas are utilized to ensure only one SAD computation is performed for each incoming pixel. For a number of row blocks, m, and a number of column blocks, n, search areas are evenly distributed for each block with $s_m \times s_n$ pixels, formulated in

$$s_m = \left\lfloor \frac{W}{m} \right\rfloor, \quad s_n = \left\lfloor \frac{H}{n} \right\rfloor, \tag{1}$$

where W and H represent image width and image height, respectively.

The template block positions (blue) and their corresponding search areas (green) are illustrated in Figure 2. In each clock cycle, only one template block is matched with one block from its corresponding search area. As each template block will only search in its dedicated search area without intruding other regions, the whole block matching process shares only one SAD computation unit for processing the whole image, allowing m and n to be context-switched in runtime.

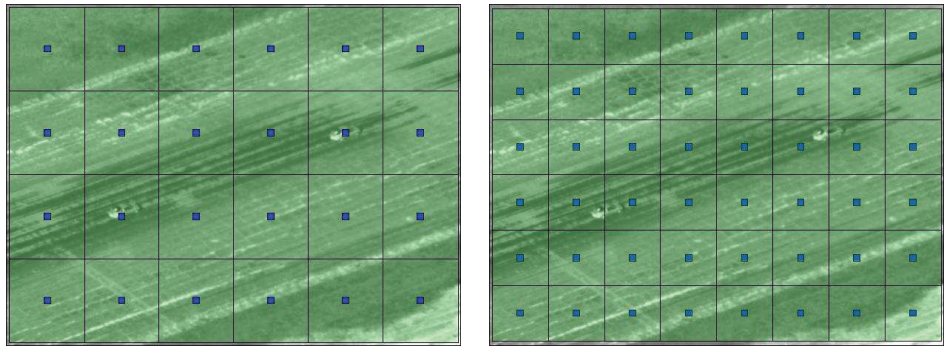
The proposed approach is able to perform different densities of area-based registration using the same hardware cost. However, higher density reduces the search areas of each block, thus limiting the flow displacement (travel distance of each point). The displacement limitations in horizontal d_m and vertical d_n are given as $d_m = \pm W/2m$ and $d_m = \pm H/2n$, respectively. As the position and movement of UAV (height, velocity, etc.) as well as frame rate of captured aerial video affect the point-to-point displacement between two successive frames, the proposed technique will produce wrong image registration result if the point-to-point displacement between frames exceeds d_m in horizontal or/and d_n in vertical.

3.2. RANSAC. After the block matching stage, a set of point pairs (point-to-point motion) from two successive frames are identified. Based on these point pairs, ego-motion estimation can be performed. As outliers (inconsistent motions) usually appear in these point pairs, RANSAC algorithm is applied to remove outliers from the data. RANSAC is an iterative algorithm to find the affine model that best describes the transformation of the two subsequent frames. Unlike the conventional RANSAC [38], this work uses an upper bound time to terminate RANSAC computation (similar to [39]) regardless of the number of iterations due to the real-time constraint as illustrated in Algorithm 1.

At each iteration, RANSAC algorithm chooses three distinct point pairs randomly as samples. Hypothesis model of affine transformation is then generated from the selected samples based on

$$\begin{bmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 \\ h_3 & h_4 & h_5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix}, \tag{2}$$

where h_i denote the parameters of the affine model to be estimated, x_i and y_i are the coordinates of chosen sample points, and x_i' and y_i' represent their corresponding point pairs.



(a) 6 × 4 blocks (b) 8 × 6 blocks

Figure 2: Positions of template blocks (blue) and search areas (green) in video frame for different densities ($m \times n$) of block matching with same hardware cost.

while time taken < upper bound time **do**
(1) Randomly select 3 distinct point pairs as samples.
(2) Generate hypothesis model (affine parameters) based on the chosen samples.
(3) Apply $T_{d,d}$ test on the hypothesis model.
(4) Calculate the fitness score of the model.
(5) Update and store best scored parameters.
end while

Algorithm 1: RANSAC algorithm.

fitness score = 0
for all data_i **do**
$\text{asub}x = \text{abs}(x_{2i} - (x_{1i} \cdot H_0 + y_{1i} \cdot H_1 + H_2))$
$\text{asub}y = \text{abs}(y_{2i} - (x_{1i} \cdot H_3 + y_{1i} \cdot H_4 + H_5))$
$\text{score} = \min((\text{asub}x^2 + \text{asub}y^2), \text{th}_{\text{dist}}^2)$
fitnessscore = fitnessscore + score
end for
Where:
Each data_i contains a point pair (x_{1i}, x_{2i}, y_{1i}, and y_{2i})
$H_0, H_1, H_2, H_3, H_4, H_5$ are affine parameters of hypothesis model.
$\text{th}_{\text{dist}}^2$ is the predefined distance threshold.

Algorithm 2: Fitness scoring in RANSAC algorithm.

$T_{d,d}$ test proposed in [40] is applied in the algorithm to speed up RANSAC computation by skipping the following steps (step (4) and (5)) if the hypothesis model is far from the truth. Fitness of the hypothesis is then evaluated and scored by fitting its parameters to all point pairs. The best hypothesis model is constantly updated in each iteration and emerges as the final result when the RANSAC is terminated upon reaching an upper bound time. As RANSAC has the least computation among overall moving target detection algorithms, it is implemented as software program with only the fitness scoring step (step (4)) being hardware accelerated. Fitness scoring is the calculation of the fitness for a hypothesis model towards all input data (point pairs from block matching), as described in Algorithm 2.

Each data is considered as an inlier if its fitting error is smaller than a predefined distance threshold, th_{dist} or vice versa. Inlier fitness score is its fitting error while outlier score is fixed to th_{dist} as a constant penalty. The total fitness score is calculated by accumulating all individual scores for each data where a perfect fit will have zero fitness score. As fitness scoring is an iterative process for all data, the number of computations increases with size of data. As RANSAC is a stochastic algorithm, it may not produce the best-fit affine model when given limited iteration.

3.3. Object Segmentation. After estimating ego-motion, the camera movement between two successive frames is to be compensated prior to object foreground detection. The previous frame is transformed and mosaic with current frame using the estimated affine parameters from RANSAC algorithm. Reverse mapping technique is applied by calculating the corresponding location in the source image based on the destination pixel location. The equation of affine transformation is shown in

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 \\ h_3 & h_4 & h_5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \quad (3)$$

where x_i and y_i are the pixel coordinates of destination image, x_i' and y_i' denote the corresponding pixel coordinates in source image, and h_i are best-fit affine parameters from RANSAC.

As the transformation may produce fractional result, nearest neighbour interpolation is utilized due to its efficiency in hardware design. The ego-motion compensation is performed pixel-by-pixel in raster scan, generating a stream of the transformed previous frame to the next process.

Frame differencing is executed on the current frame and the transformed (ego-motion compensated) previous frame by pixel-to-pixel absolute subtraction of both frames. The pixels in the resultant image are threshold with constant

Table 2: Pipeline scheduling for processing subsequent frames.

Processes	Processing frame at frame period, t_i						
	t_0	t_1	t_2	t_3	t_4	$\cdots$	t_i
Motion estimation							
(i) Block matching	F_0	$F_1 \leftarrow F_0$	$F_2 \leftarrow F_1$	$F_3 \leftarrow F_2$	$F_4 \leftarrow F_3$	$\cdots$	$F_i \leftarrow F_{i-1}$
(ii) RANSAC	—	—	$F_1 \leftarrow F_0$	$F_2 \leftarrow F_1$	$F_3 \leftarrow F_2$	$\cdots$	$F_{i-1} \leftarrow F_{i-2}$
Object segmentation							
(i) Affine transformation							
(ii) Frame differencing	—	—	—	$F_1 \leftarrow F_0$	$F_2 \leftarrow F_1$	$\cdots$	$F_{i-2} \leftarrow F_{i-3}$
(iii) Median filtering							
(iv) Morphological							

$F_i \leftarrow F_j$ is detection of moving object from jth frame to ith frame.

value of th_{fd} to produce binary image. Lower value of th_{fd} may induce more false alarm in detection while higher value causes the miss detection. Both subtraction and thresholding processes can be done as soon as two pixels for the same coordinate from these frames are obtained to yield one binary pixel for the next process. Lastly, 7×7 binary median filter and dilation processes are performed on the binary image to remove noise and improve the detected region of moving target.

3.4. Pipeline Scheduling. In order to establish a real-time moving target detection system for streaming video, proper pipeline scheduling is utilized to fully maximize the overall system throughput. The algorithm is split into several subprocesses with each hardware accelerator working on different frames independently, transferring the intermediate result from one process to another until the end of the detection cycle. Hence, the system will always produce output every time after a fixed latency. The overall process is divided into four stages of pipeline as shown in Table 2.

Due to data dependencies of the streaming algorithm, all processes must be done sequentially to produce one detection result. Block matching requires two successive video frames for computation. The first frame is streamed in for block extraction process and stored into frame buffer. Block matching is performed after the next frame is obtained with the extracted block of previous frame. RANSAC can only begin its computation after block matching has finished processing on the entire frame. Lastly, two original frames (F_{i-2} and F_{i-3}) are read from frame buffer for object segmentation to produce the final result. Object segmentation computation can be performed in stream without further frame buffering. The overall pipeline processing of the streaming system has four frames' latency. Hence, at least four frames (F_{i-3} to F_i) must be stored in frame buffer at all time for a complete moving target detection process.

4. Proposed Moving Target Detection SoC

The moving target detection SoC is developed and prototyped in Terasic DE2-115 board with Altera Cyclone IV FPGA device. The system consists of hardware/software codesign of the algorithm of where the hardware computation is executed in dedicated accelerator coded in Verilog Hardware Description Language (HDL) while software program is performed using a soft-core Nios II processor with SDRAM as software memory. The system architecture of the proposed moving target detection SoC is illustrated in Figure 3.

Camera interface handles the image acquisition tasks to provide the raw image for processing, while VGA interface manages video displaying task. Apart from being a software memory, part of SDRAM is also reserved as video display buffer. Thus, Direct Memory Access (DMA) technique is applied to read and write the displaying frame in SDRAM to ensure the high throughput image transfer.

As multiple frames are required at the same time to detect moving target, frame buffer is required to temporarily store the frames for processing. Hence, SRAM is utilized as frame buffer due to its low latency access. Since most computations are performed in the dedicated hardware, Nios II handles only RANSAC process (except fitness scoring step as described in Section 3.2) and auxiliary firmware controls. USB controller is included in the SoC to enable data transfer with USB mass storage device for verification and debugging purposes. In addition, embedded operating system (Nios2-linux) is booted in the system to provide file system and drivers support.

The real-time video is streamed directly into the moving target detector for processing. Both Nios II and hardware accelerator modules compute the result as a hardware/software codesign system and transfer the output frame to SDRAM via DMA. VGA interface constantly reads and displays the output frame in SDRAM. All operations are able to be performed in real-time, attaining a 30 fps moving target detection system.

4.1. Moving Target Detection Hardware Accelerator. The hardware architecture of the moving target detector is shown in Figure 4. It is composed of motion estimation core, object segmentation core, frame grabber, and other interfaces. The overall moving target detection is performed according to the following sequences:

(1) Frame grabber receives the input video stream and stores four most recent frames (F_{i-3} to F_i) into frame buffer through its interface. At the same time frame

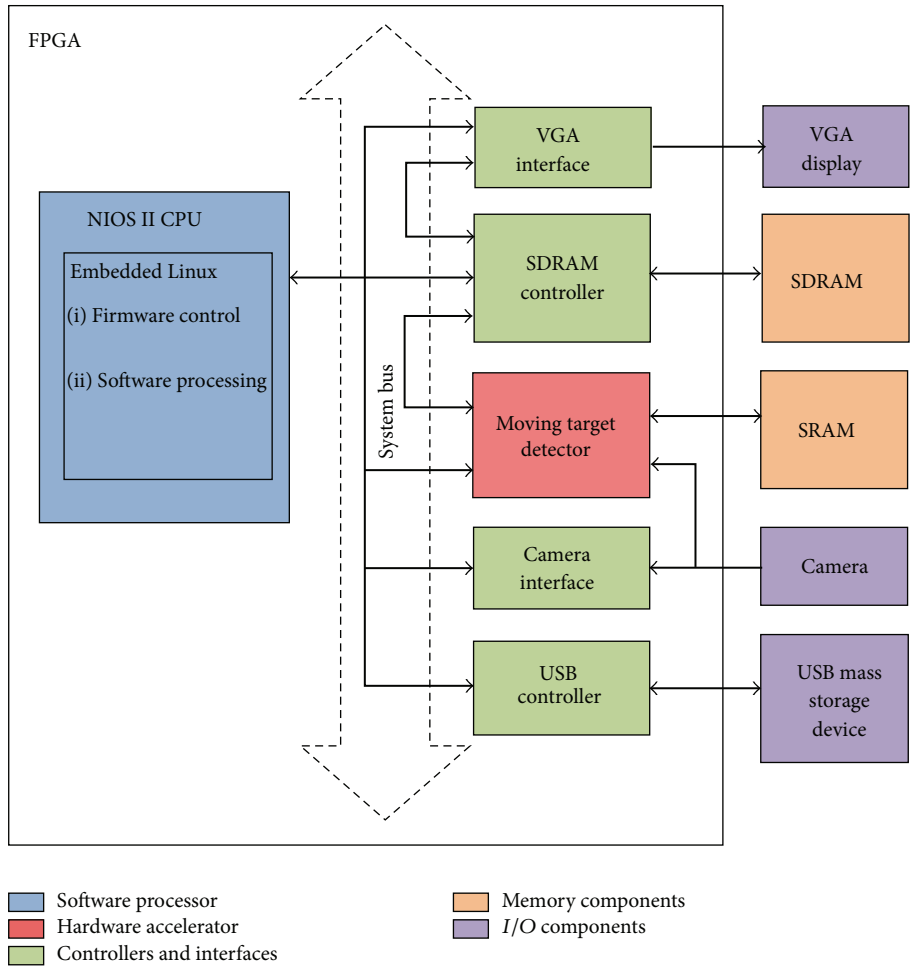


Figure 3: System architecture of moving target detection.

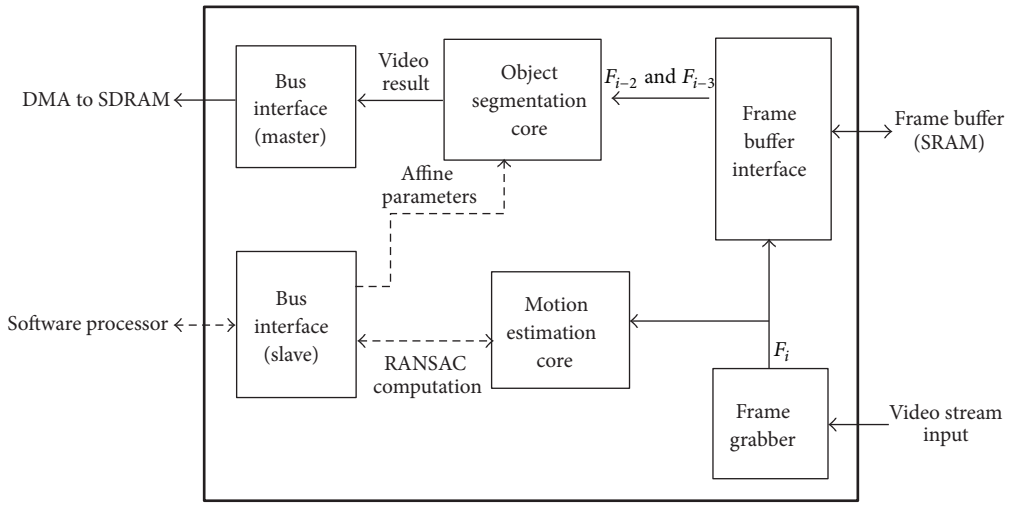


Figure 4: Hardware architecture of moving target detector.

grabber also provides the current frame (F_i) to motion estimation core.

(2) Motion estimation core performs block matching and RANSAC computation. Since RANSAC is computed in both hardware and software, software processor is constantly accessing this core via system bus interface to calculate the affine parameters.

(3) After RANSAC, the affine parameters are transferred from software to object segmentation core. Two previous frames (F_{i-2} and F_{i-3}) are read from the frame buffer by object segmentation core for processing.

(4) Several processes involving affine transformation, frame differencing, median filter, and dilation are then performed on both frames, resulting in the detected moving target.

(5) Lastly, the bus interface (master) provides DMA access for object segmentation core to transfer the end result into SDRAM for displaying and verification purposes.

As the frame buffer (SRAM) is a single port 16-bit memory, frame grabber concatenates two neighbouring 8-bit greyscale pixels to store in one memory location. Since frame grabber and object segmentation core share the frame buffer to write and read frames, respectively, frame buffer interface provides priority arbitration and gives frame grabber the highest priority, granting every write request. However, frame buffer may be busy for a couple of clock cycles due to read operation of SRAM by other modules; a small FIFO with depth of 4 is utilized in frame grabber to temporarily buffer the incoming image pixels.

4.2. Motion Estimation Hardware Accelerator. Motion estimation core consists of block matching and RANSAC hardware accelerators. Since RANSAC requires the entire data of point pairs provided by block matching to begin its computation, additional buffers are needed to temporarily store the corresponding point pairs for every two subsequent frames. The hardware architecture for motion estimation process is shown in Figure 5.

To enable high throughput data (point pairs) sharing for both block matching and RANSAC, double buffering technique is applied by using two buffers (Buffer 1 and Buffer 2) as data storage. For any instance, one buffer is written by block matching while the other is used for computation by RANSAC. Buffer controller swaps the roles of these two buffers for each incoming new frame, therefore ensuring both processes to be pipelined by reading and writing on each buffer subsequently. Buffer swapping is initiated at each completion of block matching modules while RANSAC is performed during the time gap between each swap and is terminated before the next swap.

4.2.1. Block Matching Hardware Accelerator. Figure 7 shows the architecture of the proposed block matching hardware accelerator, performing template blocks extraction from one frame and matching of these template blocks in their corresponding search areas from next frame. The overall process can be completed in stream to yield the point-to-point motion (point pairs) of two subsequent frames without buffering an entire frame.

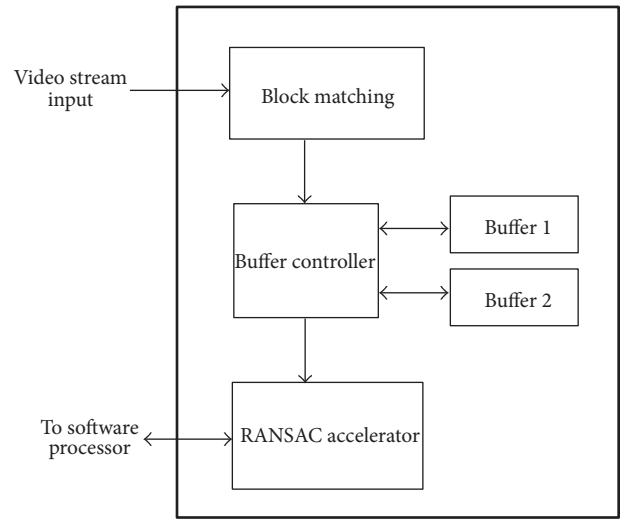
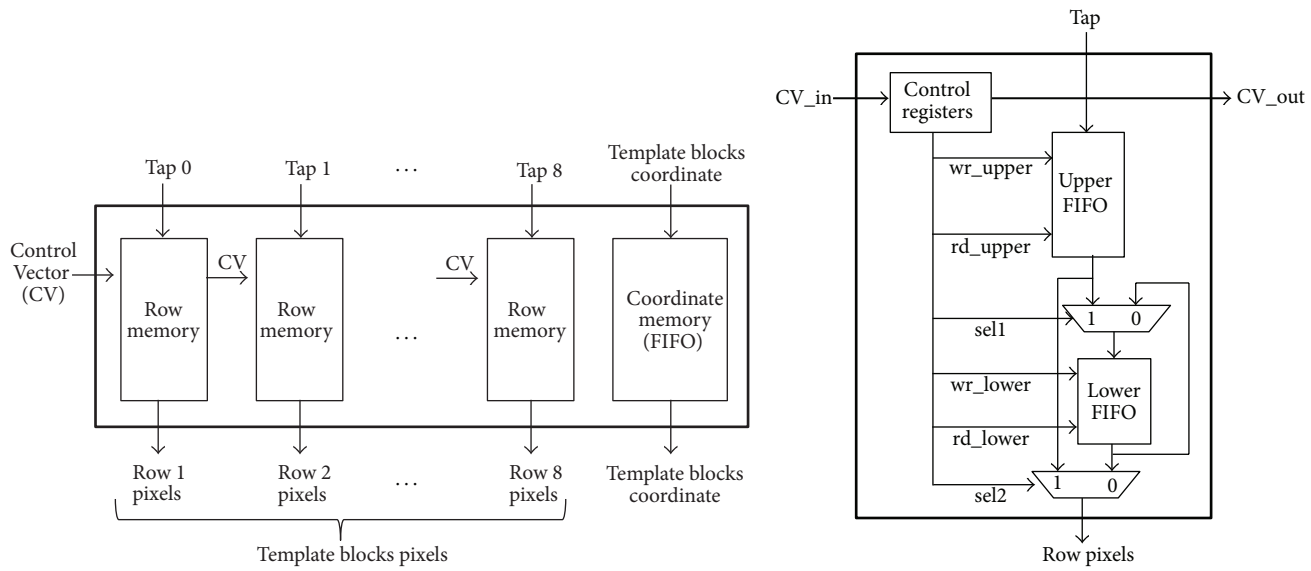


Figure 5: Hardware architecture of motion estimation core.

As 9×9 block size is utilized in block matching, a 9-tap line buffer is designed in such a way that 9×9 pixels of moving window can be obtained in every clock cycle. These 9×9 pixels are shared for both block extraction and matching processes and are read one by one in pipeline from the line buffer at each valid cycle, resulting in a total of 81 cycles to obtain a complete window.

The block extractor keeps track of the coordinate of current pixel in video stream as a reference for extraction process. Template blocks from incoming frames are extracted and stored temporarily into block memory. As each block is extracted line-by-line in raster scan, block memory is divided into nine-row memories as illustrated in Figure 6(a) with each of which being used to store one pixel row in template blocks. When video stream reaches the block position, each pixel row is loaded into each row memory from the corresponding tap of the line buffer. Block coordinates are also stored in a separate FIFO to keep track of its position.

Since only one SAD processor is used for matching $m \times n$ blocks as mentioned in Section 3.1, the template block has to be swapped according to the corresponding search area during raster scan. Hence, row memory is constructed with two FIFOs, upper and lower FIFO as illustrated in Figure 6(b), to enable block swapping during matching process. Template blocks are stored into upper FIFO during extraction process. During matching process, each line of raster scan enters eight different search areas to match eight different template blocks, respectively. Hence, one row of template blocks is cached in lower FIFO and is repeatedly used until the end of their search areas (reaching next row of search areas). Upon reaching each new row of search areas, template blocks in lower FIFO are replaced with new row of template blocks from upper FIFO. At the last line of raster scan, the lower FIFO is flushed to prevent overflow.



(a) Block memory consisting of nine-row memories

(b) Row memory contains an upper FIFO and lower FIFO

Figure 6: Block memory architecture for storing template blocks.

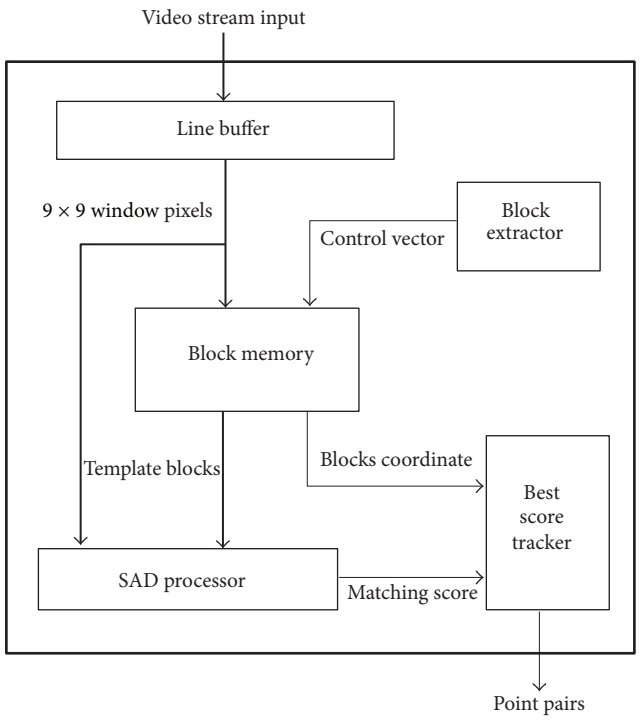


Figure 7: Stream-oriented hardware architecture of block matching.

Table 3: Control Vector (CV) for different read and write operations of block memory.

Position of raster scan	Write, upper	Read, upper	Write, lower	Read, lower	sel1	sel2
Entering template block position	1	x	x	x	x	x
Entering first search area row	x	1	1	0	1	1
Entering next search area row	x	1	1	1	1	1
Reentering same search area row	x	0	1	1	0	0
Leaving last search area row	x	0	0	1	0	0

In order to efficiently extract and match all blocks, different Control Vector (CV) as illustrated in Table 3 is sent to perform different reading and writing operations in block memory based on the current position in raster scan. Both reads and writes are independent of each other and are able to be executed at the same time. Pixels are processed one by one in 81 cycles to complete a window. Both writing and reading processes require 9 cycles for each row memory, passing CV from the first row memory to the next row memory until the end to complete a 81-pixel write or read operation of a template block.

SAD processor performs the correlation of the template blocks from previous frame with all possible blocks from current frame according to the search area. Extracted block pixels are read from block memory, while window pixels in search areas are provided from the taps of the line buffer. The total number of required PEs is the total number of pixels in a window. The process is pipelined such that each pixel is computed in each PE as soon as it is obtained from the line buffer. Matching score of each window can be obtained in every cycle after a fixed latency.

Lastly, the best score tracker constantly stores and updates the best matching score for each template block within its corresponding search area. The matching score is compared

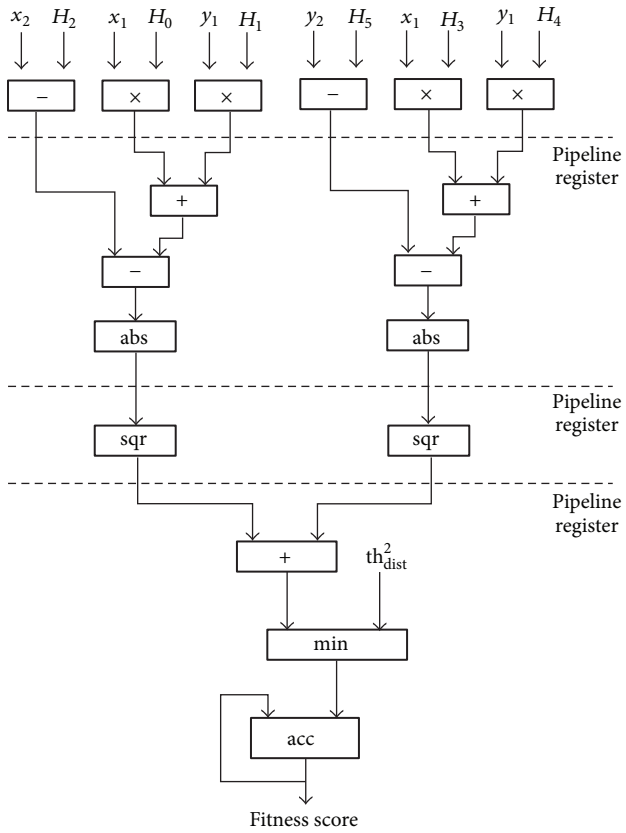


Figure 8: Hardware datapath of fitness scoring in RANSAC accelerator.

Table 4: Fixed point precision of fitness scoring inputs.

Parameter	Number of bits		Number range
	Integer	Fraction	
x_1, y_1, x_2, y_2	11	0	[−1024, 1024)
H_0, H_1, H_3, H_4	4	12	[−8, 8)
H_2, H_5	11	5	[−1024, 1024)

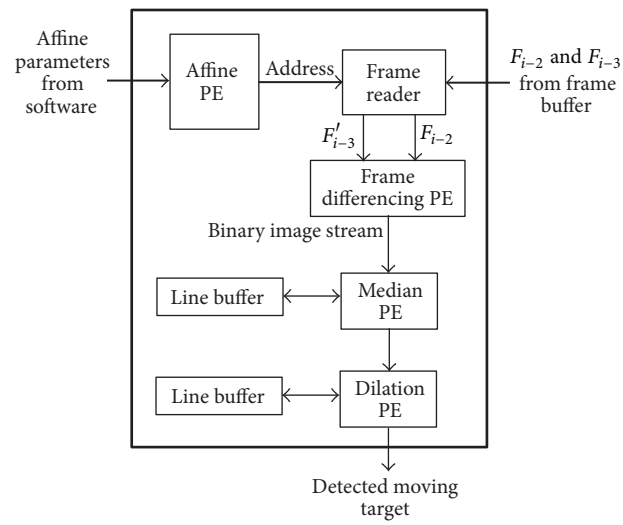


Figure 9: Hardware architecture for object segmentation.

among the same search area and the coordinates of the best-scored blocks are preserved. At the end of each search area, the coordinates of the best pairs (template blocks and their best-scored blocks) are sent to RANSAC module for next processing. Hence, the proposed block matching hardware is able to produce point-to-point motion (point pairs) of every two successive frames in streaming video at line rate.

4.2.2. RANSAC Hardware Accelerator. RANSAC hardware design in [39] is utilized in this work, which accelerates only fitness scoring step. As described in Algorithm 2, fitness scoring is an iterative process which performs similar computation to all data samples based on hypothesis model. Hence, this data intensive process is executed in pipelined datapath as illustrated in Figure 8. A control unit is utilized to read input data provided by block matching from buffer and stream these inputs to the datapath unit at every clock cycle.

The datapath unit utilizes three stages of pipeline with the aim of isolating multiplication processes, thus allowing faster clock rate. The first stage pipeline registers are located right after the first multiplication, while the other two stages of pipeline registers enclose the squaring processes. The individual score is accumulated in the last stage, producing total final fitness score. The accumulator is reset on each new set of hypothesis. Thus, the total number of cycles required for fitness score computation is the number of overall data plus the four-cycle latency.

Although fitness scoring could require floating point computations, the datapath unit uses suitable fixed point precision for each stage. Since Nios II is a 32-bit processor, the affine parameters in hypothesis model (H_0 to H_6) are properly scaled to different precision of 16-bit fixed points as described in Table 4 so that two affine parameters can be assigned in a single 32-bit write instruction. As this system is targeted for 640×480 pixels' video, all input coordinates (x_1, y_1, x_2, and y_2) are scaled to 11 bits.

4.3. Object Segmentation Hardware Architecture. As object segmentation can be performed in one raster scan, a stream-oriented architecture is proposed as illustrated in Figure 9. All subprocesses are executed in pipeline on the streaming video without additional frame buffering. Object segmentation process is initiated by software processor after providing the affine parameters from RANSAC to affine PE. Two frames (F_{i-2} and F_{i-3} as described in Table 2) from frame buffer (SRAM) are required to segment the moving target.

Based on the affine parameters from RANSAC, affine PE uses reverse mapping technique to find each pixel location in previous frame (F_{i-3}) using (3) and generates their addresses in frame buffer (SRAM). Frame readers fetch the previous frame (F_{i-3}) pixel-by-pixel according to the generated addresses from frame buffer, thus constructing a stream of transformed frame, which is denoted as F'_{i-3}.

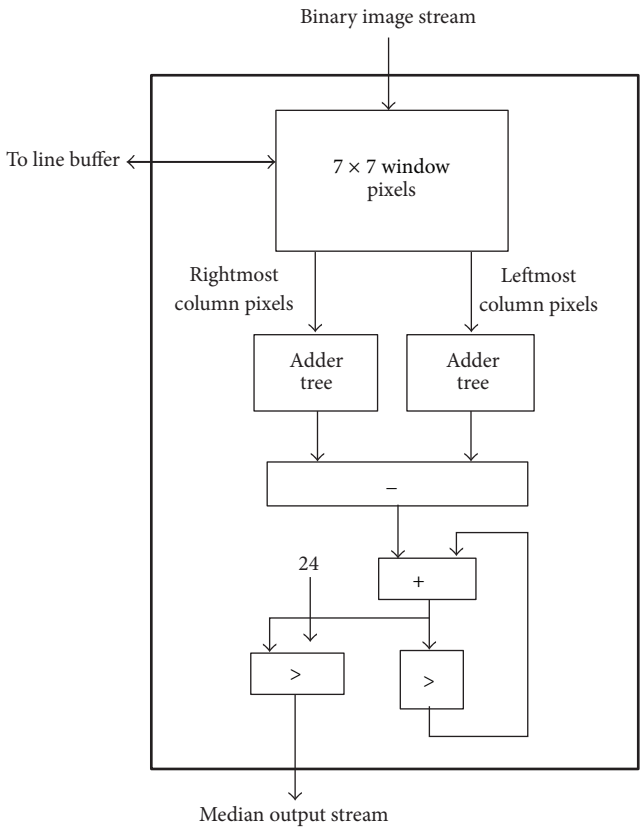


Figure 10: Hardware architecture of median PE.

By synchronizing the streams of both frames, frame differencing can be executed in pipeline as soon as one pixel from each frame is obtained. Hence, one pixel in current frame (F_{i-2}) and one pixel in transformed frame (F'_{i-3}) are fetched alternatively from their corresponding memory locations by frame reader, constructing two synchronized streams of F_{i-2} and F'_{i-3} frames. Frame differencing PE performs pixel-to-pixel absolute subtraction and thresholding on the streams. The frame differencing PE is able to compute in one cycle per pixel. A configurable threshold value, th_{fd}, is used after the subtraction, yielding a stream of binary image without buffering the whole frame.

After frame differencing the binary image is streamed into 7 × 7 median filtering. Seven lines of the image are buffered in the line buffer, providing 7 × 7 pixels window for the median PE to perform the median computation. Median computation can be performed in one clock cycle for each processing window due to short propagation delay as only binary pixels are involved. Figure 10 shows the hardware logic design of median PE.

Median filtering can be computed by counting the number of asserted (binary 1) pixels in the window. If more than half the pixels in the window (24 pixels) are asserted, the resultant pixel is "1," or "0" otherwise. Since processing window will move only one pixel to the right for each computation during raster scan, current pixel count is computed by adding the previous pixel count and rightmost column pixels in the current window while subtracting the leftmost column pixels in the previous window. Final binary output pixel is produced by thresholding the current pixel count with 24 (half of window size).

As dilation is also a 7 × 7 window-based processing, it uses similar line buffering technique as median filtering. However, only simple logical OR operation is performed on all window pixels. Due to its simplicity, dilation PE can also be computed in one clock cycle, resulting in the stream of binary image with detected region of moving targets.

5. Experimental Results

5.1. Verification of Proposed SoC. The proposed moving target detection SoC is verified in offline detection mode using the database in [41]. Test videos are 640 × 480 pixels in size and are greyscaled prior to the verification process. The test videos are transferred to the system for computation via a USB mass storage device. After performing the detection in SoC, the image results are displayed on VGA and also stored on USB drive for verification. Figure 11 shows the moving target detection result from the proposed SoC using different sample videos. The detected regions (red) are overlaid on the input frame. In most cases, the proposed SoC is able to detect the moving target in consecutive frames.

However, there are several limitations in this work. Block matching may not give a good motion estimation result if the extracted blocks do not have texture (the pixels intensity are similar). Moreover, the detected region of moving target may appear in cavity or multiple split of smaller regions as only simple frame differencing is applied in the proposed system. Additional postprocessing to produce better detected blob by merging split regions is out of the scope in this work.

As the stochastic RANSAC algorithm is terminated after a constant time step for each frame, image registration error may occur which produces incorrect ego-motion estimation. This could be mitigated by accelerating RANSAC algorithm to ensure more iterations using dedicated hardware or high performance general purpose processor.

5.2. Performance Evaluation of Detection Algorithm. The performance evaluation of the implemented detection algorithm uses the Mathematical Performance Metric in [42] that involves several parameters as follows:

(i) True positive, TP: the detected moving object.

(ii) False positive, FP: detected regions that do not correspond to any moving object.

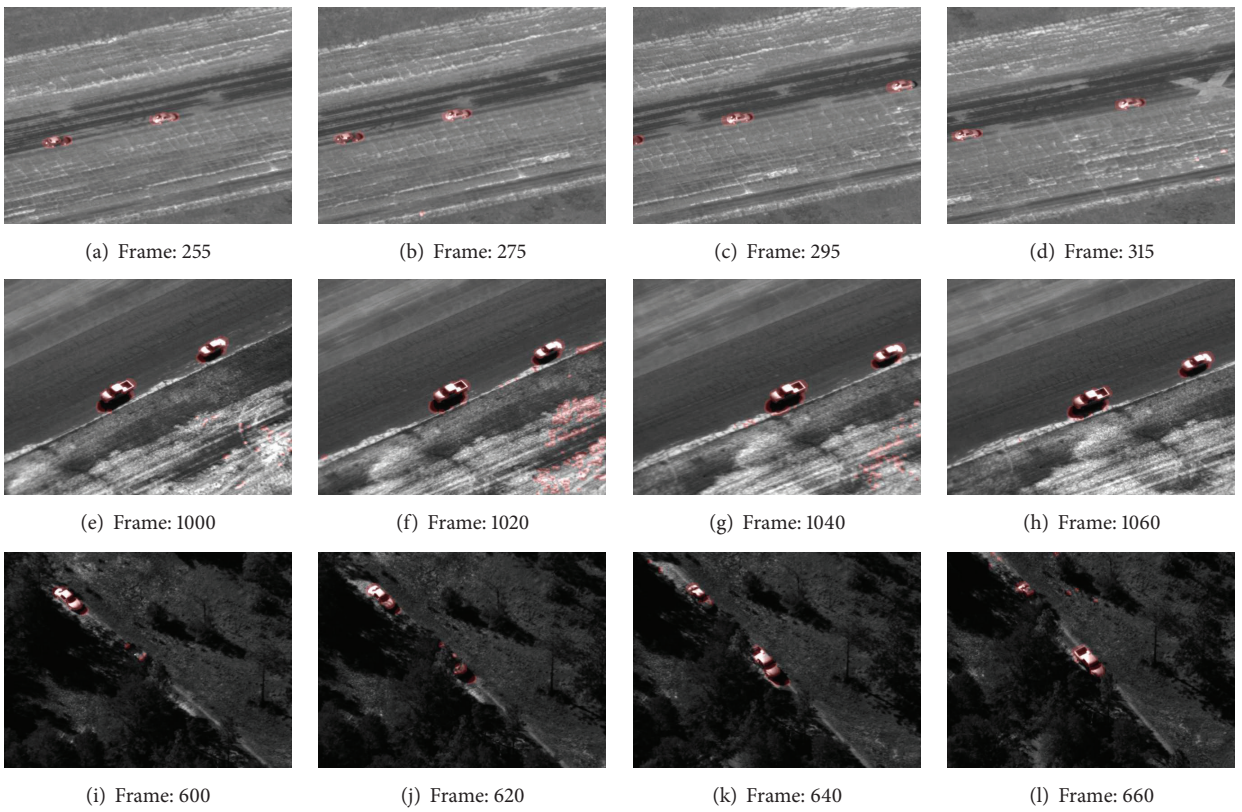
(iii) False negative, FN: the nondetected moving object.

(iv) Detection rate, DR: the ratio of TP with the combination of TP and FN, as formulated in

$$\text{DR} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4}$$

(v) False alarm rate, FAR: the ratio between FP in all positive detection, as defined in

$$\text{FAR} = \frac{\text{FP}}{\text{TP} + \text{FP}}. \tag{5}$$



(a) Frame: 255 (b) Frame: 275 (c) Frame: 295 (d) Frame: 315

(e) Frame: 1000 (f) Frame: 1020 (g) Frame: 1040 (h) Frame: 1060

(i) Frame: 600 (j) Frame: 620 (k) Frame: 640 (l) Frame: 660

Figure 11: Detected regions from the proposed moving target detection SoC on different sample videos in [41]. Video numbers (a)–(d): V3V100003_004, video numbers (e)–(h): V3V100004_003, and video numbers (i)–(l): V4V100007_017.

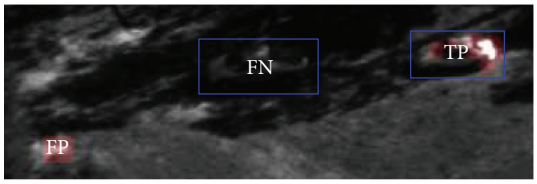


Figure 12: Evaluation of performance metrics TP, FP, and FN based on ground truth boxes (blue) and the detected region (red).

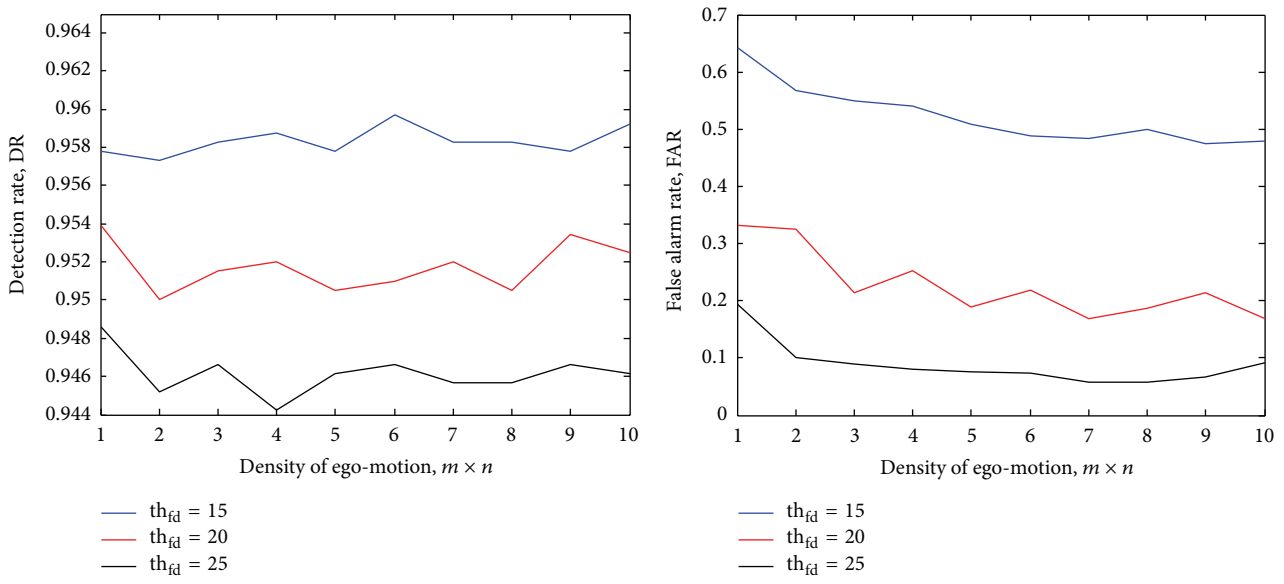
To obtain the performance metrics, ground truth regions are manually labelled in several frames of test videos. A bounding box is drawn across each moving object to indicate the ground truth region of every frame as depicted in Figure 12. A simple postprocessing is performed on the detected region by filtering out the detected region smaller than 15 pixels' width or 15 pixels' height prior to the evaluation. A detected moving object (TP) has detected regions in its bounded ground truth area, while a nondetected moving object (FN) has no detected region overlapping with its ground truth area. Detected region that does not overlapp with any ground truth region is considered as false positive (FP).

The detection performance is evaluated on different parameters configuration. The DR and FAR for 1000 test frames using different number of blocks (density in ego-motion estimation), $m \times n$, in area-based registration and frame differencing threshold, th_{fd}, are depicted in Table 5 and Figure 13.

The experiment results show that DR is almost similar for different density of ego-motion estimation but decreases with th_{fd}. Although higher density in the proposed work has lower displacement limitation, d_m and d_n as discussed in Section 3.1, most of the point-to-point displacements do not exceed the limitation due to slow UAV movement in the most frames of the test dataset. On the contrary, higher value of th_{fd} may filter out the moving object if the differences in intensity of the object pixels and background pixels are almost similar.

FAR decreases with density in ego-motion estimation due to the higher quality in image registration process but increases if most frames exceed the displacement limitation, d_m and d_n. However, false registration due to displacement limitation results in a huge blob of foreground but does not greatly increase FAR. Although higher values of th_{fd} decrease the false detection rate, they also produce smaller foreground area for all detected moving objects as pixels almost similar intensity with background will be thresholded.

5.3. Speed Comparison with Full Software Implementation. The computation speed of the proposed moving target detection SoC is compared with software computation in different platforms, including modern CPU (Intel Core i5) in desktop computer and embedded processor (ARM). Table 6 illustrates the comparison of computation frame rate and hardware



(a) DR (b) FAR

Figure 13: DR and FAR for different density in ego-motion estimation, $m \times n$, and frame differencing threshold, th_{fd}.

Table 5: Performance evaluation in terms of DR and FAR for 1000 frames using different density in ego-motion estimation, $m \times n$, and frame differencing threshold, th_{fd}.

$m \times n$	th_{fd}	DR	FAR
12	15	0.958	0.643
12	20	0.954	0.331
12	25	0.949	0.194
24	15	0.957	0.568
24	20	0.950	0.324
24	25	0.945	0.101
35	15	0.958	0.548
35	20	0.952	0.215
35	25	0.947	0.090
48	15	0.959	0.539
48	20	0.952	0.253
48	25	0.944	0.079
70	15	0.958	0.509
70	20	0.951	0.188
70	25	0.946	0.075
88	15	0.960	0.489
88	20	0.951	0.219
88	25	0.947	0.074
108	15	0.958	0.483
108	20	0.952	0.168
108	25	0.946	0.058
140	15	0.958	0.499
140	20	0.951	0.187
140	25	0.946	0.059
165	15	0.958	0.474
165	20	0.953	0.214
165	25	0.947	0.068
192	15	0.959	0.478
192	20	0.952	0.169
192	25	0.946	0.092

Table 6: Computation speed comparison of the proposed system with different software implementation using area-based and feature-based registrations.

Platform	Frequency	Registration technique	Frame rate	Hardware speed-up
Proposed SoC	100 MHz	Area-based	30	1
Intel Core i5-4210U	1.70 GHz	Area-based	4.26	7.04
		Feature-based	13.11	2.29
ARM1176JZF	700 MHz	Area-based	0.20	150
		Feature-based	0.56	53.57

speed-up between the proposed system and other software implementations using test videos in [41].

As feature-based image registration has faster computation in software implementation comparing to area-based registration, speed performance of feature-based method is also included for comparison. In feature-based implementation, features are first detected in each frame. The detected features from current frame are cross-correlated with features with previous frame while RANSAC algorithm is used to estimate the ego-motion between frames. After compensating the ego-motion, segmentation of moving object uses the same processes with the proposed system. To further optimize the software implementation in terms of speed performance, a fast feature detection algorithm [30] is utilized. As the number of features will affect the computation time in feature matching step, only 100 strongest features in each frame are selected for processing. However, the performance evaluation does not consider multithreaded software execution.

Table 7: Resources usage of the proposed moving target detection SoC.

	Logic units	Utilization (%)
Total combinational function	15161	13%
Total registers	10803	9%
Total memory bits	521054	13%
Embedded multiplier	27	5%
FPGA device	Altera Cyclone IV	

Based on experimental result, the speed performance of the proposed moving target detection SoC surpasses optimized software computation by 2.29 times and 53.57 times compared with implementations in modern CPU and embedded CPU, respectively. The software computation (RANSAC) in HW/SW codesign of the proposed system creates speed bottleneck, thus limiting the maximum throughput to 30 fps. The processing frame rate of the proposed system can be further improved by using fully dedicated hardware.

5.4. Resource Utilization. The overall hardware resources utilization of the complete system is illustrated in Table 7. This prototype of real-time moving object detection system utilizes only less than 20 percent of total resources in Altera Cyclone IV FPGA device. As the proposed system uses off-chip memory components for frame buffering, FPGA on-chip memory is utilized only for line buffering in streaming process (e.g., block matching and median filtering) and storing intermediate results (e.g., point pairs after block matching). Thus, the low resource usage of the proposed system provides abundant hardware space for other processes such as target tracking or classification to be developed in future.

6. Conclusions

Moving target detection is a crucial step in most computer vision problem especially for UAV applications. On-chip detection without the need of real-time video transmission to ground will provide immense benefit to diverse applications such as military, surveillance, and resource exploration. In order to perform this complex embedded video processing on-chip, FPGA-based system is desirable due to the potential parallelism of the algorithm.

This paper proposed a moving target detection system using FPGA to enable autonomous UAV which is able to perform the computer vision algorithm on the flying platform. The proposed system is prototyped using Altera Cyclone IV FPGA device on Terasic DE2-115 development board mounted with a TRDB-D5M camera. This system is developed as a HW/SW codesign using dedicated hardware with Nios II software processor (booted with embedded Linux) running at 100 MHz clock rate. As stream-oriented hardware with pipeline processing is utilized, the proposed system achieves real-time capability with 30 frames per second processing speed on 640×480 live video. Experimental result shows that the proposed SoC performs 2.29 times and 53.57 times faster than optimized software computation on modern desktop computer (Intel Core i5) and embedded processor (ARM). In addition, the proposed moving target detection uses only less than 20 percent of total resources in the FPGA device, allowing other hardware accelerators to be implemented in future.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The authors would like to express their gratitude to Universiti Teknologi Malaysia (UTM) and the Ministry of Science, Technology and Innovation (MOSTI), Malaysia, for supporting this research work under research Grants 01-01-06-SF1197 and 01-01-06-SF1229.

References

[1] A. Ahmed, M. Nagai, C. Tianen, and R. Shibasaki, "Uav based monitoring system and object detection technique development for a disaster area," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 37, pp. 373–377, 2008.

[2] B. Coifman, M. McCord, R. Mishalani, M. Iswalt, and Y. Ji, "Roadway traffic monitoring from an unmanned aerial vehicle," *IEE Proceedings-Intelligent Transport Systems*, vol. 153, no. 1, pp. 11–20, 2006.

[3] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "Survey of unmanned aerial vehicles (uavs) for traffic monitoring," in *Handbook of Unmanned Aerial Vehicles*, pp. 2643–2666, Springer, 2015.

[4] K. Nordberg, P. Doherty, G. Farnebäck et al., "Vision for a UAV helicopter," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS '02), Workshop on Aerial Robotics*, pp. 29–34, Lausanne, Switzerland, October 2002.

[5] D. Zamalieva and A. Yilmaz, "Background subtraction for the moving camera: a geometric approach," *Computer Vision and Image Understanding*, vol. 127, pp. 73–85, 2014.

[6] M. Genovese and E. Napoli, "ASIC and FPGA implementation of the Gaussian mixture model algorithm for real-time segmentation of high definition video," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 537–547, 2014.

[7] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "An embedded real-time surveillance system: implementation and evaluation," *Journal of Signal Processing Systems*, vol. 52, no. 1, pp. 75–94, 2008.

[8] H. Jiang, H. Ardö, and V. Öwall, "A hardware architecture for real-time video segmentation utilizing memory reduction techniques," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 2, pp. 226–236, 2009.

[9] M. Genovese and E. Napoli, "FPGA-based architecture for real time segmentation and denoising of HD video," *Journal of Real-Time Image Processing*, vol. 8, no. 4, pp. 389–401, 2013.

[10] A. Lopez-Bravo, J. Diaz-Carmona, A. Ramirez-Agundis, A. Padilla-Medina, and J. Prado-Olivarez, "FPGA-based video system for real time moving object detection," in *Proceedings*

of the 23rd International Conference on Electronics, Communications and Computing (CONIELECOMP '13), pp. 92–97, IEEE, Cholula, Mexico, March 2013.

[11] T. Kryjak, M. Komorkiewicz, and M. Gorgon, "Real-time moving object detection for video surveillance system in FPGA," in *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP '11)*, pp. 1–8, IEEE, Tampere, Finland, November 2011.

[12] A. Mittal and D. Huttenlocher, "Scene modeling for wide area surveillance and image synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 160–167, IEEE, June 2000.

[13] A. Price, J. Pyke, D. Ashiri, and T. Cornall, "Real time object detection for an unmanned aerial vehicle using an FPGA based vision system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, pp. 2854–2859, IEEE, Orlando, Fla, USA, May 2006.

[14] G. J. García, C. A. Jara, J. Pomares, A. Alabdo, L. M. Poggi, and F. Torres, "A survey on FPGA-based sensor systems: towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing," *Sensors*, vol. 14, no. 4, pp. 6247–6278, 2014.

[15] S. Ali and M. Shah, "Cocoa: tracking in aerial imagery," in *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications III*, vol. 6209 of *Proceedings of SPIE*, Orlando, Fla, USA, April 2006.

[16] J. Xiao, C. Yang, F. Han, and H. Cheng, "Vehicle and person tracking in aerial videos," in *Multimodal Technologies for Perception of Humans*, pp. 203–214, Springer, 2008.

[17] W. Yu, X. Yu, P. Zhang, and J. Zhou, "A new framework of moving target detection and tracking for uav video application," in *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science*, vol. 37, Beijing, China, 2008.

[18] V. Reilly, H. Idrees, and M. Shah, "Detection and tracking of large number of targets in wide area surveillance," in *Computer Vision—ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part III*, vol. 6313 of *Lecture Notes in Computer Science*, pp. 186–199, Springer, Berlin, Germany, 2010.

[19] J. Wang, Y. Zhang, J. Lu, and W. Xu, "A framework for moving target detection, recognition and tracking in UAV videos," in *Affective Computing and Intelligent Interaction*, vol. 137 of *Advances in Intelligent and Soft Computing*, pp. 69–76, Springer, Berlin, Germany, 2012.

[20] S. A. Cheraghi and U. U. Sheikh, "Moving object detection using image registration for a moving camera platform," in *Proceedings of the IEEE International Conference on Control System, Computing and Engineering (ICCSCE '12)*, pp. 355–359, IEEE, Penang, Malaysia, November 2012.

[21] Y. Zhang, X. Tong, T. Yang, and W. Ma, "Multi-model estimation based moving object detection for aerial video," *Sensors*, vol. 15, no. 4, pp. 8214–8231, 2015.

[22] Q. Yu and G. Medioni, "A GPU-based implementation of motion detection from a moving platform," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR '08)*, pp. 1–6, Anchorage, Alaska, USA, June 2008.

[23] A. Laika, J. Paul, C. Claus, W. Stechele, A. E. S. Auf, and E. Maehle, "FPGA-based real-time moving object detection for walking robots," in *Proceedings of the 8th IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR '10)*, pp. 1–8, IEEE, Bremen, Germany, July 2010.

[24] B. Tippetts, S. Fowers, K. Lillywhite, D.-J. Lee, and J. Archibald, "FPGA implementation of a feature detection and tracking algorithm for real-time applications," in *Advances in Visual Computing*, pp. 682–691, Springer, 2007.

[25] K. May and N. Krouglicof, "Moving target detection for sense and avoid using regional phase correlation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '13)*, pp. 4767–4772, IEEE, Karlsruhe, Germany, May 2013.

[26] M. E. Angelopoulou and C.-S. Bouganis, "Vision-based egomotion estimation on FPGA for unmanned aerial vehicle navigation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 6, pp. 1070–1083, 2014.

[27] I. M. El-Emary and M. M. A. El-Kareem, "On the application of genetic algorithms in finger prints registration," *World Applied Sciences Journal*, vol. 5, no. 3, pp. 276–281, 2008.

[28] A. A. Goshtasby, *2-D and 3-D Image Registration: for Medical, Remote Sensing, and Industrial Applications*, John Wiley & Sons, New York, NY, USA, 2005.

[29] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, vol. 15, pp. 147–151, 1988.

[30] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision—ECCV 2006*, pp. 430–443, Springer, 2006.

[31] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: speeded up robust features," in *Computer Vision—ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., vol. 3951 of *Lecture Notes in Computer Science*, pp. 404–417, Springer, 2006.

[32] G. R. Rodríguez-Canosa, S. Thomas, J. del Cerro, A. Barrientos, and B. MacDonald, "A real-time method to detect and track moving objects (DATMO) from unmanned aerial vehicles (UAVs) using a single camera," *Remote Sensing*, vol. 4, no. 4, pp. 1090–1111, 2012.

[33] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 2, pp. 148–157, 1993.

[34] R. Li, B. Zeng, and M. L. Liou, "New three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp. 438–442, 1994.

[35] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 313–317, 1996.

[36] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 287–290, 2000.

[37] L. De Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1309–1316, 1989.

[38] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[39] J. W. Tang, N. Shaikh-Husin, and U. U. Sheikh, "FPGA implementation of RANSAC algorithm for real-time image geometry

estimation," in *Proceedings of the 11th IEEE Student Conference on Research and Development (SCOReD '13)*, pp. 290–294, IEEE, Putrajaya, Malaysia, December 2013.

[40] O. Chum and J. Matas, "Randomized ransac with $T_{d,d}$ test," in *Proceedings of the British Machine Vision Conference*, vol. 2, pp. 448–457, September 2002.

[41] DARPA, SDMS Public Web Site, 2003, https://www.sdms.afrl.af.mil.

[42] A. F. M. S. Saif, A. S. Prabuwono, and Z. R. Mahayuddin, "Motion analysis for moving object detection from UAV aerial images: a review," in *Proceedings of the International Conference on Informatics, Electronics and Vision (ICIEV '14)*, pp. 1–6, IEEE, Dhaka, Bangladesh, May 2014.