RISE: An Automated Framework for Real-Time Intelligent Video Surveillance on FPGA

BITA DARVISH ROUHANI, University of California San Diego
AZALIA MIRHOSEINI, Google Brain
FARINAZ KOUSHANFAR, University of California San Diego

This paper proposes RISE, an automated Reconfigurable framework for real-time background subtraction applied to Intelligent video SurveillancE. RISE is devised with a new streaming-based methodology that adaptively learns/updates a corresponding dictionary matrix from background pixels as new video frames are captured over time. This dictionary is used to highlight the foreground information in each video frame. A key characteristic of RISE is that it adaptively adjusts its dictionary for diverse lighting conditions and varying camera distances by continuously updating the corresponding dictionary. We evaluate RISE on natural-scene vehicle images of different backgrounds and ambient illuminations. To facilitate automation, we provide an accompanying API that can be used to deploy RISE on FPGA-based system-on-chip platforms. We prototype RISE for end-to-end deployment of three widely-adopted image processing tasks used in intelligent transportation systems: License Plate Recognition (LPR), image denoising/reconstruction, and principal component analysis. Our evaluations demonstrate up to 87-fold higher throughput per energy unit compared to the prior-art software solution executed on ARM Cortex-A15 embedded platform.

CCS Concepts: •**Computing methodologies** →**Machine learning;** •**Computer systems organization** →*Embedded systems; Real-time systems;*

Additional Key Words and Phrases: Intelligent video surveillance, Data streaming, Background subtraction, License plate recognition, Reconfigurable computing.

ACM Reference format:
Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. 2017. RISE: An Automated Framework for Real-Time Intelligent Video Surveillance on FPGA. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2017), 18 pages.
DOI: https://doi.org/10.1145/3126549

1 INTRODUCTION

Background subtraction is a challenging task in various video surveillance applications including but not limited to traffic monitoring, vandalism deterrence, and suspicious object detection [1]. In background subtraction algorithms, the primary step is to learn a reference model (a.k.a., a dictionary matrix) to effectively represent the background scene in a video stream. Several factors might affect the background scene in a real-world setting, making it necessary to design dynamic streaming algorithms/tools in order to handle new structural trends that might appear in the input

This work was supported in parts by the Office of Naval Research grant (ONR N00014-17-1-2500) and National Science Foundation TrustHub grant (1649423).
This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2017 and appears as part of the ESWEEK-TECS special issue.

DOI: https://doi.org/10.1145/3126549

data. Examples of such effects include sudden or gradual illumination changes, the presence of shadows, background repetitive movements (e.g., waving trees), and moving objects.

A number of background subtraction realization on GPUs [2], ASICs [3], and FPGAs [4, 5] have been reported in the literature. The existing solutions, however, either (i) have adopted a static approach in a sense that they require having access to a pre-defined set of referenced background images [4, 5]; thereby, they cannot adapt to the changing geometry of the data due to the varying backgrounds and ambient illuminations. Or (ii) require learning a large over-complete dictionary matrix to comply with the uncontrolled environments in the background pixels, e.g., [1, 6]. The use of over-complete dictionaries, in turn, bounds the applicability of these works on constrained System-on-Chip (SoC) platforms due to the high memory footprint and the computational budget requirement for online processing using large data matrices.

We propose RISE, a novel automated computing framework for FPGA-based real-time background subtraction applied to intelligent video surveillance. RISE is devised with a new adaptive, and streaming-based methodology that enables real-time monitoring of persistent and transient objects in uncontrolled environments. The new method is scalable and well-suited for embedded settings with severe resource constraints. The resource limitation can be categorized in terms of the available memory, processing time, and/or energy (e.g., battery life). RISE takes the stream of video frames captured by surveillance cameras as its input and highlights the foreground content existing in each frame to subsequently reduce the objection detection/tracking workload such that it complies with the pertinent resource provisioning/constraints.

As the stream of surveillance video frames comes in, RISE learns/updates a fixed-size dictionary matrix as a subsample of the input pixels. The dictionary samples are carefully selected inline with the data arrival such that it captures the varying background pixels. Our framework uses a greedy sparse-coding routine called Orthogonal Matching Pursuit (OMP) to adaptively transform the original video frames to a new set of frames (images) by eliminating the background pixels based on the learned dictionary. Note that processing the foreground content restrict the computations to the most informative regions of the input images. As we empirically corroborate in this paper, this computational reduction, in turn, makes it practical to realize object detection/tracking applications on resource-constrained SoC platforms available on today's surveillance cameras. As such, it evades the need to transfer large video collections to a control station for further processing by enabling on-chip data analysis.

RISE is designed based on a HW/SW co-design approach. To facilitate automation and adaptation of the proposed framework, we provide a set of open-source libraries/tools that can be used for automated deployment of an arbitrary surveillance application using FPGA platforms. The explicit contributions of this paper are as follows:

- Developing RISE, an automated and customized set of algorithms and hardware-accelerated tools that enable real-time background subtraction on generic images applied to intelligent video surveillance applications.
- Enabling adaptivity to various data settings. RISE maintains a fixed-size dictionary which is formed by scalably sub-sampling the input data. As the background varies over time, the dictionary samples are dynamically updated to account for evolving data structure.
- Facilitating hardware customization. RISE takes physical limitations such as memory budget, and real-time processing requirement into consideration and creates the best dictionary that meets the aforementioned constraints.
- Performing extensive evaluations on various types of images and video frames. The results demonstrate both algorithmic practicality and system performance efficiency of the proposed framework.

2 PRELIMINARIES

In this section, we briefly discuss the streaming data modeling in constrained embedded settings (Section 2.1), and the Orthogonal Matching Pursuit algorithm (Section 2.2).

2.1 Streaming Data Analysis

Streaming data analysis refers to a set of algorithms and tools that enable online processing of data streams in settings with severe physical constraints (e.g., available memory, energy, and/or computational budget). To deal with the resource constraints, in streaming scenarios, the computation is performed using a *sketch* (approximate summary) of the input data. The data sketch should be acquired inline with the data arrival in a pass-efficient manner (usually through only one pass). It is worth noting that the use of streaming algorithms/tools enables on-chip real-time reasoning from data content collected on sensor nodes without having to transfer the raw data to an off-site processing unit.

2.2 Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) is a greedy algorithm for solving compressed sensing and sparse approximation problems [7, 8]. For a given dictionary matrix (D), OMP algorithm aims to sparsely reconstruct (approximate) the input data measurement (Y) based on the pertinent dictionary. Let us denote the maximum desired sparsity level with k and the target error threshold to ϵ. Algorithm 1 outlines the pseudocode of OMP routine [8]. As shown, performing the OMP routine for a given signal requires iterative execution of three main steps: (i) finding the best matching sample in the dictionary matrix D (Line 4 of Algorithm 1), (ii) least-square (LS) optimization (Line 5 of Algorithm 1), and (iii) residual update (Line 6 of Algorithm 1). In our pseudocode, D_j represents the j^{th} column of matrix D and D_Λ is the subset of D consisting of the columns defined in the set Λ. The OMP algorithm terminates when the number of non-zero elements in the output coefficient vector is more than the sparsity level k or the L_2-norm of the residual vector ($\|r\|_2$) is less than the user-defined threshold ϵ.

Algorithm 1 OMP Algorithm

Inputs: Dictionary D, input sample Y, maximum sparsity level k, and desired error threshold ϵ.

Output: Coefficient vector V.

1: $\Lambda^0 \leftarrow []$
2: $i \leftarrow 0$
3: $r^0 \leftarrow Y$
4: **while** ($i \leq k$ and $\|r^i\|_2 > \epsilon$) **do**
5: $\quad \Lambda \leftarrow \Lambda \cup argmax_j | < r^{i-1}, D_j > |$
6: $\quad V^i \leftarrow argmin_v \|r^{i-1} - D_{\Lambda^i} V\|_2^{\ 2}$
7: $\quad r^i \leftarrow r^{i-1} - D_{\Lambda^i} V^i$
8: $\quad i \leftarrow i + 1$
end while

As shown, OMP is an iterative computationally expensive algorithm with a complex data flow. RISE is devised with a scalable hardware-accelerated realization of OMP routine using FPGA

platform. The implementation of OMP routine on FPGA, in turn, enables low-power and real-time computation in embedded settings with limited energy and resource provisioning (e.g., memory capacity).

3 RISE FRAMEWORK

Figure 1 demonstrates the high-level block diagram of RISE framework. As shown, RISE consists of three main building blocks for on-chip realization of object detection/tracking applications: (i) dictionary learning unit (Section 3.1), (ii) background subtraction unit (Section 3.2), and (iii) user-defined post-processing unit (Section 3.3). RISE is devised with an streaming-based data analysis methodology to enable real-time image processing for intelligent surveillance applications. The proposed method computes the sketch of a video stream by learning a set of dynamic dictionaries to best represent the input data. The data sketch is then used in RISE framework to perform a user-specific learning application such as object tracking/localization.

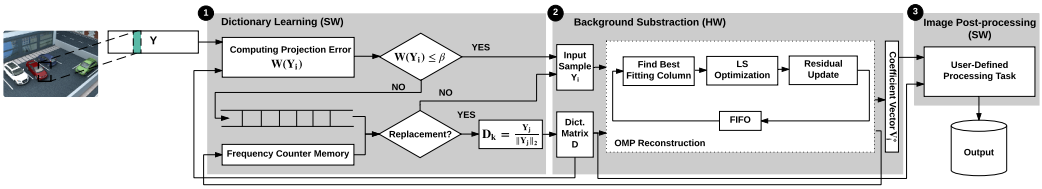


Fig. 1. High-level block diagram of RISE framework. RISE takes the stream of input data as its input and gradually learns the underlying data structure to perform background subtraction. The framework consists of three main building blocks for real-time on-chip analysis of streaming data: dictionary learning performed in software, background subtraction accelerated by hardware, and user-defined image post-processing executed in software.

3.1 Adaptive Dictionary Learning

Many modern data collections are either low-rank or lie on a union of lower dimensional subspaces [9, 10]. RISE leverages this data property to effectively identify the foreground outliers in a video stream data by learning a low-rank dictionary matrix that is continuously updated as data evolves over time. For a given stream of video data, RISE first converts each frame into a series of patches that contain the values of neighboring pixels as illustrated in Figure 2. The size of each data patch and the desired overlap between two successive patches are algorithmic parameters that are tuned with respect to the pertinent memory availability and runtime budget in RISE framework.

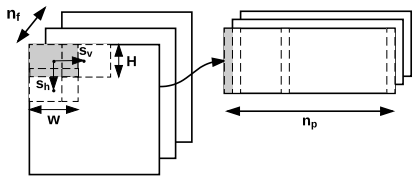


Fig. 2. Image to patch conversion: each input image is mapped to a series of overlapping data patches to be processed in the dictionary learning and background subtraction units. A similar approach is performed to transform the processed data patches back to the input image size after object detection.

In RISE framework, the dictionary matrix is initiated as an empty set. The dictionary is then gradually filled up with the input data patches that best presents the background content. To do so, for each arriving data patch, RISE computes a projection error based on the current values of the dictionary matrix (D) as follows:

$$\mathbf{W(Y_i)} = \frac{\|\mathbf{D(D^TD)^{-1}D^TY_i - Y_i}\|_2}{\|\mathbf{Y_i}\|_2}, \tag{1}$$

where the term $D(D^TD)^{-1}D^T$ indicates the projection space spanned by the column space of dictionary matrix D.

The projection error $W(.)$ is an indicator that shows whether the existing content in the input data patch can be well represented within a user-defined approximation error (β) or not. If the newly arrived data patch does not deviate significantly from the model spanned by the selected dictionary samples, it proceeds to the next step for background subtraction (Section 3.2). Otherwise, the input data patch is temporarily added to the end of a fixed-sized queue to be included in the dictionary matrix. Let us denote the maximum size of the pertinent dictionary matrix with s_d. If the number of current samples in D is less than s_d, the input data patch will be normalized and added to the dictionary. Else way, a copy of the input data is stored in the queue to be later replaced with a non-frequently used sample in the dictionary and the patch is redirected to the background subtraction unit (See Figure 1 for the data flow in the dictionary learning unit).

To bound the memory footprint of the learned dictionary and adjust for varying ambient backgrounds, RISE carefully selects and replaces old dictionary samples that are inadequate to represent the background contents appeared in the newly added video frames. To do so, RISE keeps track of the number of times each dictionary sample is used in the background subtraction unit over a certain period of time. This parameter is indicated by f_d, where d is the index number of each dictionary sample. If f_d is less than a threshold α (e.g., $\alpha = 100$) for a dictionary sample, RISE replaces that sample with a new patch of data retrieved from the queue. This is because, if a dictionary sample is not used frequently in the system it either belongs to the background samples in previous video frames that are not applicable anymore due to the change of background scene or ambient conditions, or it is mixed with the foreground pixels. Algorithm 2 shows the pseudocode of RISE framework. The "while loop" continues until the system is interrupted by the user. The indicated boxes in Algorithm 2 corresponds to the building blocks illustrated in Figure 1.

3.2 Background Subtraction

Background subtraction can be cast as a sparse signal reconstruction problem. In this step, pixels that significantly differ from the model spanned by the selected dictionary samples are regarded as foreground. In other words, the background patches in each video frame Y_{n_f} can be sparsely represented as the linear combination of a few samples in the learned dictionary. RISE solves the following objective function using orthogonal matching pursuit to find the sparse representation of the data (V_{n_f}) with respect to the dictionary matrix (D):

$$minimize_{V^*_{n_f}} \|Y_{n_f} - DV_{n_f}\|_2 \; s.t. \; \|V^*_{n_f}\|_0 \leq k. \tag{2}$$

Here, $\|.\|_0$ denotes the number of non-zeros in the pertinent vector, and $\|.\|_2$ indicates the L_2-norm. In particular, the p-norm of a vector x with $p \geq 1$ is computed as:

$$\|x\|_p = (\sum_{j=1}^{n} |x(j)|^p)^{1/p}. \tag{3}$$

Algorithm 2 RISE Framework

Inputs: Video stream A, user-defined thresholds (α, β, ϵ), target sparsity level k, dictionary size s_d, and Morphological statistics $Stat_M$.
Output: Dictionary Matrix D, and desired learning output L.

$D \leftarrow []$
$f_d \leftarrow [0]_{s_d \times 1}$
$n_f \leftarrow 0$
while *(true)* **do**
  $Y_{n_f} \leftarrow image2patch(A_{n_f})$ ❶
  $D \leftarrow DictionaryLearning(Y_{n_f}, D, s_d, f_d, \alpha, \beta)$
  $(V^*_{n_f}, f_d) \leftarrow BackgroundSubtraction(D, Y_{n_f}, k, \epsilon)$ ❷
  $L \leftarrow PostProcessing(D, V^*_{n_f}, Stat_M)$ ❸
  $n_f \leftarrow n_f + 1$
  Pause until the next frame is available
end while

OMP is a well-known routine for solving compressed sensing and sparse approximation problems [8]. It takes a dictionary D and a sample Y_{n_f} as inputs and iteratively approximates the sparse representation of the sample by adding the best fitting element in every iteration (Section 2). Parameter k controls the total number of non-zeros per column of V (a.k.a., sparsity level). Suppose the dictionary matrix D contains a set of background patches, thus the background in each frame Y_{n_f} can be reconstructed using $DV^*_{n_f}$, where $V^*_{n_f}$ is the optimal solution per Eq. (2). As such, the foreground information can be represented as a sparse matrix by computing

$$Y_{n_f} - DV^*_{n_f}. \tag{4}$$

Depending on the user-specific learning task, RISE either uses the foreground information ($Y_{n_f} - DV^*_{n_f}$) or the sparse image reconstruction ($DV^*_{n_f}$) for post-processing of the input data stream. For instance, in object detection applications such as license plate recognition, the foreground information should be used for localization of the items of interest. However, in a particular classification task such as hyper-spectral imaging, analyzing the background dynamic content is of special importance.

3.3 Image Post-Processing

In this step, the sparse representation of each image (the foreground/background information extracted through previous steps) are analyzed to perform a user specific learning task. For instance, to implement an image classification experiment using principal component analysis the user can replace the original dense input image with its projected representative (DV^*) and do the computations on the projected data. Note that unlike the original input data stream, the transformed data (V^*) possesses a sparse structure which, in turn, translates to a fewer number of required floating-point operations (FLOPs) and less memory footprint.

In another scenario, a user might leverage the extracted foreground content for object localization/tracking purposes. To do so, a locally adaptive thresholding method can be performed to transform the gray-scale background subtracted image ($Y_{n_f} - DV^*_{n_f}$) into a binary image based on the local statistics of the neighborhood pixels such as range and variance [11]. In our evaluation for licence plate recognition, we used the following criteria to compute the threshold corresponding to

each patch of data:

$$T(x,y) = m(x,y) + [1 + \gamma(\frac{\sigma(x,y)}{R} - 1)], \tag{5}$$

where $m(x,y)$ and $\sigma(x,y)$ are the local sample mean and variance respectively. $R = 256$, and $\gamma = 0.5$ are used in our practical design experiment in Section 7.

Following the image binarization, Connected Component Analysis (CCA) might be applied (depending on the user-defined task) to determine the group of pixels that are probably related to one another. CCA is developed based on the assumption that pixels in a connected component (object) share similar pixel intensity values and are connected with their adjacent pixels. Once all groups have been determined, each pixel is labeled with a value according to the component to which it was assigned. To detect/track a particular object in the newly labeled set of images, one can make use of simple morphological statistics ($Stat_M$) such as the aspect ratio (a.k.a., eccentricity), and object orientation. For instance, to locate license plates in a surveillance video our evaluations shows that using $1.5 \leq aspectratio \leq 4.5$ and $10^\circ \leq orientation \leq 60^\circ$ are effective criteria to successfully locate license plate regions in an image. We emphasize that our proposed framework is generic and can be used for a variety of surveillance applications other than ones evaluated in this paper.

4 HARDWARE IMPLEMENTATION

Execution of OMP algorithm is the key computational bottleneck in RISE framework. RISE is devised based on a HW/SW co-design approach. The dictionary learning and data post-processing analysis are performed using an embedded general purpose processor while background subtraction is accelerated with FPGA. We provide a scalable implementation of OMP routine on FPGA to enable low-power and real-time analysis of streaming data. Figure 3 illustrates the high-level schematic depiction of RISE's OMP kernel. In our experiments, we leverage Peripheral Component Interconnect Express (PCIe) port to load the data to the FPGA board and write back the results to the general purpose processor hosting the FPGA. All computations in RISE framework are performed using IEEE 754 single precision floating-point format. Floating-point representation enables RISE to be readily adapted in the realization of different learning tasks without requiring the user to fine-tune the implementation corresponding to the target application.

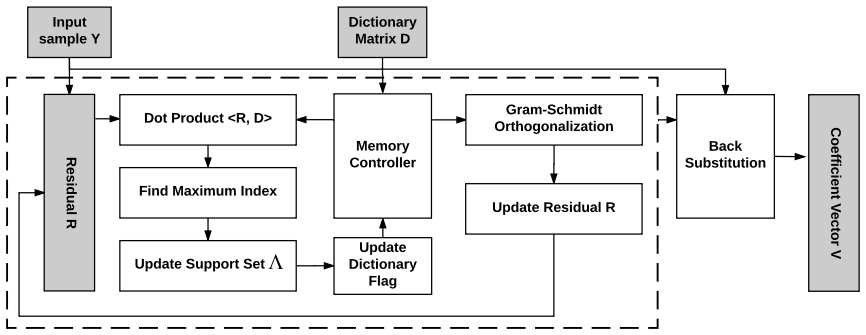


Fig. 3. Block diagram of RISE OMP kernel. The computational blocks surrounded by the dashed box iteratively work for k (target sparsity level) epochs and the result is passed to the back substitution unit to compute the coefficient vector V.

Solving the LS optimization within OMP algorithm involves a variety of operations (e.g., matrix inversion) with complex data flow. RISE leverages Gram-Schmidt orthogonalization to reduce the hardware complexity of OMP algorithm by gradually forming an orthogonal matrix U and an upper triangular matrix Z as suggested in [12]. Algorithm 3 outlines the Gram-Schmidt orthogonalization method [13, 14]. Using the Gram-Schmidt methodology, the residual (line 6 of Algorithm 1) is computed as:

$$R_i \leftarrow R_i U_i U_i^T R_i, \tag{6}$$

where R_i is the residual in iteration i, U_i denotes the current values of matrix U, and U_i^T is the transpose of matrix U_i. The final solution (V) is calculated by performing back substitution to find the inversion of the matrix Z and compute:

$$V = Z^{-1} U^T Y_{n_f}. \tag{7}$$

To make our notation explicit, for a patch of input data (Y_{n_f}) of size ($m \times 1$) and a dictionary matrix (D) of size ($s_d \times m$), the reconstruction vector V is of size $s_d \times 1$ and the square matrices U and Z are of size $s_d \times s_d$. The user-specific sparsity level k should be less than or equal to number of dictionary samples $k \leq s_d$.

Algorithm 3 Incremental Gram-Schmidt orthogonalization

Inputs: New column D_{Λ^i}, last iteration U^{i-1}, Z^{i-1}.

Output: U^i and Z^i.

1: $Z^i \leftarrow \begin{pmatrix} Z^{i-1} & 0 \\ 0 & 0 \end{pmatrix}$

2: $E^i \leftarrow D_{\Lambda^i}$

3: **for** j = 1,...,i-1 **do**

4: $\quad Z^i_{ji} \leftarrow (U^{i-1})^T_j E^i$

5: $\quad E^i \leftarrow E^i - Z^i_{ji} U^{i-1}_j$

end for

6: $Z^i_{ii} \leftarrow \sqrt{\|E^i\|_2{}^2}$

7: $U^i \leftarrow [U^{i-1}, \frac{E^i}{Z^i_{ii}}]$

Figure 4 illustrates the structure of the back substitution unit in the OMP kernel. Starting from the last row index of the upper-triangular matrix Z, each element of the vector V can be uniquely recovered by solving a set of linear equations. The Processing Element (PE) in Figure 4 is a multiply-add accumulator in which each element of vector V is computed as follows:

$$V_i = \frac{C_i - \Sigma_{j=i+1}^{k} Z_{ij} V_j}{Zii}. \tag{8}$$

Here, k is the desired maximum sparsity level, and C denotes the matrix product of U^T and Y_{n_f}.

RISE leverages two sets of optimization for implementation of the OMP kernel by exploiting the data and algorithmic level parallelism: **(i)** The OMP algorithm mainly consists of matrix-vector and matrix-matrix multiplications. Such operations, require efficient execution of dot product between two vectors. RISE is devised with a tree-based scheduler to pipeline the design and perform

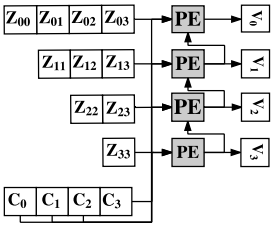


Fig. 4. Back substitution unit in RISE framework. The processing element corresponds to a recursive multiply-add accumulator used to find vector V in equation $V = Z^{-1}C$ without explicitly computing the inverse of the upper-triangular matrix Z.

matrix-based computations. The proposed approach reduces the Initiation Interval (II) in computing matrix-vector multiplication to only 1 clock cycle. We use cyclic interleaving to store the input data and the intermediate variables in OMP routine. The use of cyclic interleaving enables multiple load/store access to the memory block in each clock cycle. Assuming dual-port memory blocks, the unrolling memory factor in RISE framework is equivalent to the number of floating-point adders/multipliers that can be instantiated within the confine of the Digital Signal Processing (DSP) resources of the target FPGA. Figure 5 demonstrates the use of tree-based reduction for computing the sum value of a vector. Here, a temporary vector B is employed to store the intermediate results.

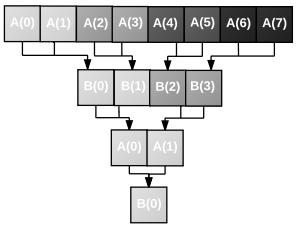


Fig. 5. Tree-based vector reduction. The use of tree-based adder enables effective pipelining of the design and reducing the II of dot product operations to only one clock cycle.

(ii) The use of block RAM is desirable on FPGA platforms due to their fast access time. However, the number of block memories on FPGAs is highly limited; thereby, it is necessary to optimize the number of utilized block RAMs in the design. The memory footprint of OMP algorithm is dominated by the memory requirement for storing the dictionary matrix D and the intermediate matrix U. RISE reuses the same set of block memories initially assigned to dictionary matrix D to store the newly added columns of matrix U at each OMP iteration. This memory reuse is possible due to the fact that the updated residual at the end of each iteration is made orthogonal to the selected dictionary samples. As such, none of the columns of matrix D would be selected twice during one call of the OMP algorithm. RISE keeps track of the already selected dictionary samples in a binary vector of size s_d. In evaluating line 4 of Algorithm 1, the binary flag is used to mask the indexes of matrix D that have been selected through the previous OMP iteration.

Figure 6 illustrates the inner structure of the OMP realization in RISE framework. In RISE framework, multiple OMP kernels can work in parallel to increase the throughput of the underlying application. RISE is devised with a global coordinator unit that includes the memory interface and a control module to manage the pertinent OMP kernels. For each newly arrived data patch, the

coordinator in RISE framework looks for the availability of the OMP kernels to assign the input data patch to an idle kernel for further processing. The control unit has also the responsibility of reading out and sending back the results to the general purpose processor hosting the FPGA.

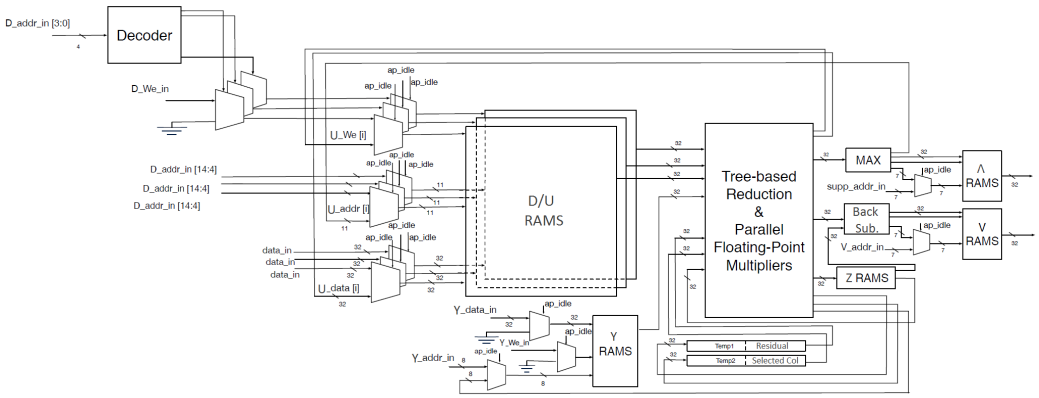


Fig. 6. OMP architecture in RISE framework. Cyclic interleaving is used to store data matrices. RISE leverages a tree-based reduction module to effectively batch the computations and pipeline the design accordingly. The same set of block memories is reused to store matrices D and U. This memory reuse reduces the underlying memory footprint by almost a factor of 2 compared to the direct mapping of OMP algorithm to FPGA without any drop in the accuracy.

5 HARDWARE SETTING AND RESULTS

We use Xilinx Virtex-6-XC6VLX240T FPGA as our primary hardware accelerator. The FPGA is programmed with a speed grade of -1 and works at the $100MHz$ clock frequency. A quad-core $2.3GHz$ ARM Cortex-A15 CPU running on the Linux 4 Tegra OS is employed for the software-based realization of RISE (used for comparison purposes). We adopt Xilinx standard IP cores for single precision floating-point operations and leverage PCIe library provided by [15] to transfer data between the host and FPGA. Xilinx ISE 14.6 is used to synthesize, place, and route the pertinent Verilog code to program the FPGA. In our prototype, four OMP kernels can be run in parallel within the confine of the pertinent resource provisioning to evaluate data patches containing less than or equal to 256 pixels.

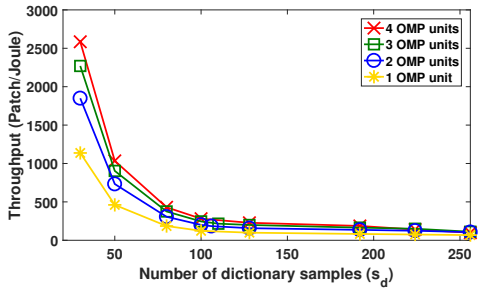


Fig. 7. RISE's worst-case throughput per energy unit as a function of dictionary size s_d. In this experiment, the data patch size is 32×8 and k is set equal to the value of s_d.

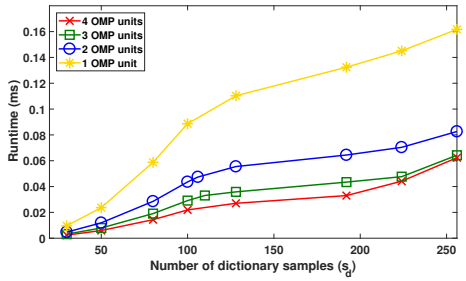


Fig. 8. RISE's average runtime per data patch as a function of dictionary size s_d. In this experiment, the data patch size is 32×8 and k is set equal to the value of s_d.

Figure 7 shows RISE's throughput per energy unit as a function of dictionary size (s_d). In this experiment, the patch size is (32×8), ϵ is 10^{-20}, and k is equal to s_d. We set $k = s_d$ to let k be as large as the dictionary size in order to evaluate the worst-case throughput. The FPGA power is simulated by Xpower analyzer tools in Xilinx ISE which accounts for both leakage and dynamic power [16]. The corresponding average runtime per input data patch in the same experimental setting is illustrated in the Figure 8. As shown in Figures 7 and 8, there is a trade-off between the system throughput and the underlying dictionary size s_d. RISE tunes the pertinent dictionary size accordingly to fit the target application runtime requirement.

The main resource bottleneck in the realization of RISE framework is the available on-chip block RAM memories on FPGA. Figure 9 shows the BRAM resource utilization for deployment of one OMP kernel with respect to various sizes of the dictionary matrix D. The DSP, Sliced Register, and Look-Up-Table (LUT) utilization remain almost the same for different values of s_d. This is because, RISE is designed to fully leverage the available computational budget (e.g., DSP units) to parallelize the underlying computations and maximize system throughput.

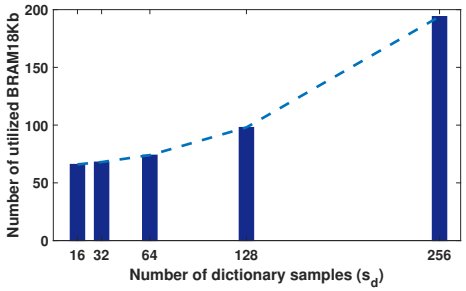


Fig. 9. BRAM resource utilization per OMP kernel for different dictionary sizes (s_d). The total available block RAM memory is $14.97Mb$ on the target FPGA platform.

Sparsity level k is a key tunable parameter in RISE framework. On the one hand, a large value of k enables the representation of the image content with more details (accuracy). This accuracy is at the cost of a higher computational workload since more OMP iterations should be performed to process each data patch. On the other hand, a very small value of k can degrade the system performance by eliminating most of the images' foreground content. Figure 10 illustrates RISE throughput per energy unit versus the target sparsity level k. In this experiment, the dictionary

size is set to $s_d = 128$ and each input data patch includes 32×8 pixels. The reported throughput corresponds to using four OMP kernels in parallel. RISE hardware implementation results in a milder slope for the larger values of the sparsity level k. This is because at each iteration of the OMP algorithm RISE restricts its search space (Line 5 of Algorithm 1) to those columns of the dictionary matrix that have not been selected during the previous OMP epochs. As such, the last OMP iterations incur less computational workload compared to the preliminary ones.

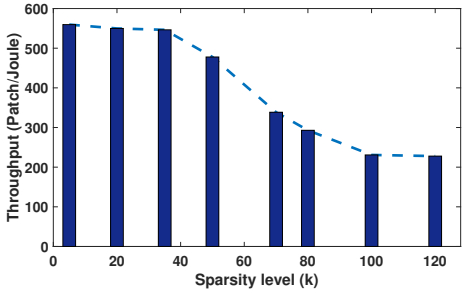


Fig. 10. RISE's throughput per energy unit as a function of sparsity level k. In this experiment, the dictionary size is $s_d = 128$, the data patch size is 32×8, and 4 OMP kernels are used in parallel.

The processing time in RISE framework is dominated by the OMP computation. Figure 11 shows the total runtime of RISE as a function of input patch size. In this experiment, the sparsity level k is set to be 50% of the data patch size and the number of samples in the dictionary is set to $s_d = 128$. The reported runtime in Figure 11 is the averaged time over 1000 samples in RISE framework.

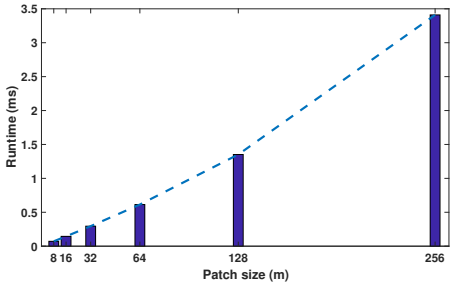


Fig. 11. RISE's total processing time as a function of data patch size. In this experiment, the dictionary size (s_d) is 128 and k is set to half of the patch size at each data point.

Video data streams are often represented as a $m \times n_p$ matrices, where n_p is the number of data patches and m is the corresponding patch size. In streaming scenarios, m is assumed to be constant within each application while n_p can be arbitrarily large as data evolves over time. As shown in Figure 12, the total latency in RISE framework is a linear function of the number of processed input data patches (n_p) which empirically collaborate the scalability of RISE HW/SW co-design approach. In this experiment, each input data patch includes 32×8 pixels and the dictionary size is set to two different values of 64 and 128.

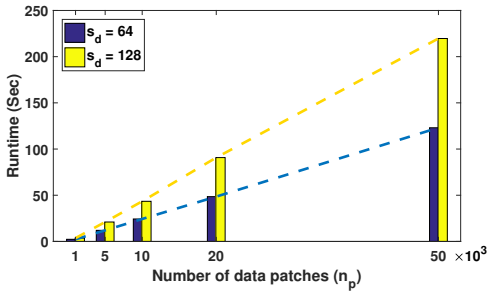


Fig. 12. RISE total processing time is linear in terms of the number of processed samples. In this experiment, the input patch size is 32×8 and the dictionary size (s_d) takes two different values of 64 and 128.

6 RISE LIBRARY

To accelerate computations and enable analyzing live video streams in a timely manner, RISE takes advantage of a HW/SW co-design approach. It leverages the high computing power of FPGAs to provide a high-throughput and low-power system. The interface of RISE framework is embedded in C++. Users can use a common C++ Integrated Development Environment (IDE) such as Visual Studio to interact with RISE framework. RISE's interface enables users to easily set their desired algorithmic parameters such as morphological statistics ($Stat_M$), and pertinent thresholds (α and β) to locate different objects in the video frames. In our prototype design, we leverage PCIe to transfer data back and forth between the FPGA and the general purpose processor hosting the FPGA. The PCIe interface can be replaced by any other data transfer link such as Ethernet depending on the application. Our realization of RISE framework is available on [17].

7 PRACTICAL DESIGN EXPERIENCE

To corroborate RISE's practicability and efficiency, we have considered three benchmark applications:

(i) License Plate Recognition (LPR): LPR is an image processing technique used to identify, track, and monitor vehicles by their license plates. This technology is used in various security and traffic applications, such as the access control, automatic congestion charge systems, or identification of dangerous drivers [18–21].

(ii) Image reconstruction/denoising: Image reconstruction/denoising is a key pre-processing step in deployment of various computer vision task [22]. We consider image denoising using dynamic dictionaries as one of the potential practical design experiences of RISE framework. In this experiment, we use a set of light-filed data acquired from [23] consisting of 2500 samples each of which includes 1600 pixels. We then solve Eq. (2) to find the best representative (DV^*) for the input data patches. In our evaluations, a Gaussian noise is added to a random subset of input data and the dictionary samples are selected by sequential streaming of input patches.

(iii) Principal Component Analysis (PCA): We have considered PCA analysis of hyperspectral images as our third practical design experiment. A hyperspectral image is a sequence of images from across the electromagnetic spectrum that is captured by hundreds of detectors embedded on satellites to classifying different materials such as soil, oil, or water included in an image. Hyperspectral imaging has a wide range of applications in earth-based and planetary explorations, geo-sensing, and beyond. However, the large size of hyperspectral datasets limits the applicability

Table 1. RISE throughput per energy unit ($\frac{frame}{sec \times watt}$) as a function of sparsity level k. In this experiment, the dictionary size s_d is 256 and data patch size is set to (32×8) with an overlap of (8×2) pixels.

Frame Size	$k = 8$		$k = 16$		$k = 32$		$k = 64$		$k = 128$	
	SW [9]	RISE	SW [9]	RISE	SW [9]	RISE	SW [9]	RISE	SW [9]	RISE
64×64	28.24	31.74	12.98	31.12	4.49	31.11	0.98	20.88	0.23	12.80
128×128	5.37	8.49	2.47	8.31	0.85	8.27	0.19	5.59	0.04	3.42
256×256	1.34	2.15	0.62	2.13	0.21	2.11	0.04	1.42	0.01	0.87

of this technique, especially for scenarios where online evaluation of a large number of samples should be performed using limited resources. In this experience, we demonstrate how RISE can be used for on-chip real-time analysis of hyperspectral data (e.g., [24]) using principal component analysis. Such analysis, in turn, enables local classification of these images using SoC components of the satellites while evading the requirements to transfer large amounts of data to the earth stations.

Table 1 details the physical performance of processing different-sized images as a function of sparsity level k. RISE streams time-series video frames and processes them sequentially. As such, the overall runtime of our framework is a linear function of the number of processed frames per video. RISE achieves up to 87-fold higher throughput per energy unit compared to a highly-optimized software solution for the OMP algorithm [9]. In this experiment, the dictionary size s_d is 256 and the patch size is set to (32×8) with an overlap of (8×2) pixels. We emphasize that the patch size, number of overlap pixels, and sparsity level are all tunable parameters that can be easily changed through RISE API to meet different runtime budget or memory constraint.

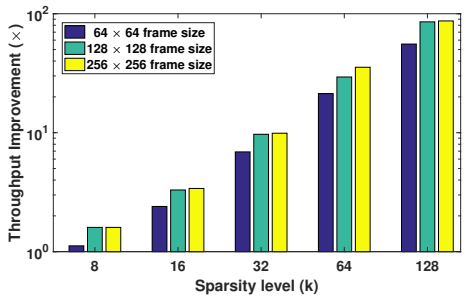
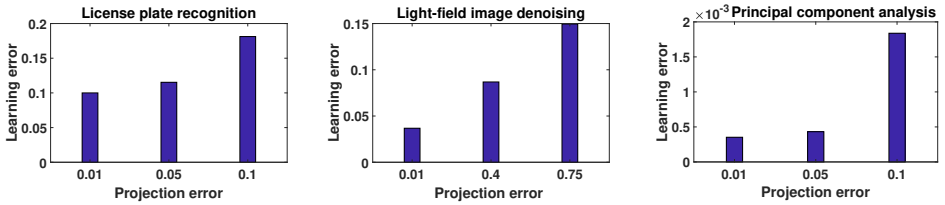


Fig. 13. The throughput per energy unit ($\frac{frame}{sec \times watt}$) improvement as a function of sparsity level for different input frame sizes. The y axis is shown in logarithmic scale. The patch size, in this experiment, is set to (32×8) with an overlap of (8×2) pixels. The dictionary size s_d is equal to 256.

Figure 13 summarizes the performance improvement achieved by RISE framework compared to the platform-customized state-of-the-art OMP deployment on an embedded CPU processor [9]. The improvement is reported in terms of the number of image frames that can be processed in each energy unit ($\frac{frame}{sec \times watt}$). The y axis in Figure 13 is shown in logarithmic scale. As illustrated, RISE gains a higher improvement for larger values of sparsity level k. This improvement is due to the domain-specific optimization of our hardware-accelerated solution (e.g., tree-based adder, memory management, etc.) as discussed in Section 4.

7.1 Error Analysis

Figure 14 shows the impact of data projection error threshold (ϵ) on the ultimate learning error in each application. As shown, although higher data projection error (lower k values) can result in meaningful performance improvements, they may not significantly affect the learning error. The reported error for image reconstruction/denoising and PCA analysis is computed as $\frac{\|Y-DV^*\|_2^2}{\|Y\|_2}$, where Y is the original noiseless ground-truth data and DV^* is the approximated solution of Eq. (2). The reported error for the LPR benchmark corresponds to the localization error for license plate detection within an image.



(a) (b) (c)

Fig. 14. The impact of data transformation error (ϵ) on the latent learning error in (a) license plate recognition, (b) image denoising/reconstruction, and (c) principal component analysis. In this experiment, the number of dictionary samples s_d is 256 and the sparsity level k is set to the value of s_d to let the projection error threshold ϵ determine the termination of OMP algorithm.

Figure 15 shows the output of RISE's background subtraction unit for an example image in which multiple cars are located near one the other in a parking lot. The dataset is acquired from [25]. The use of RISE framework evades the requirement to send over the raw RGB images to a control station. Therefore, it significantly reduces the communication delay and enables efficient execution of various object detection algorithms on a binary image. The binary image only contains the foreground information that has been highlighted using RISE's adaptive background subtraction technique.

We emphasize that RISE is a generic computing framework for adaptive background subtraction in uncontrolled environments. It simplifies feature extraction with custom codes, enabling users to hand off a lot of those decisions to the dictionary learning process. RISE throughput (up to $20fps$) is sufficient enough to perform real-time video processing as most of the commercial surveillance cameras ($> 70\%$) are operating at the rate of $10fps$ or less [26]. The high throughput of RISE framework, in turn, evades the requirement to buffer the video streams on-chip or transfer (send) the high volume of the original dense data to a control station.

(a) Input image frame [25] (b) Background subtraction output (c) Localized objects from image b

Fig. 15. Example output of RISE's framework for the deployment of an LPR system. In this experiment, the sparsity level k is set to 16, dictionary size (s_d) is 256, and input patch size is 32×8.

8 RELATED WORK

Compressive Sensing Background Subtraction. Several research studies have been focused on the use of compressive sensing and sparse-coding techniques such as the greedy OMP routine to perform background subtraction for video data collections [1, 6]. These works mainly rely on the use of a large over-complete dictionary matrix which bounds their applicability to be employed on constrained SoC platforms. RISE overcomes this limitation by proposing an adaptive dictionary learning approach that incurs a fixed low memory footprint while adapting to the dynamic nature of the underlying surveillance task. The proposed approach is computationally less expensive due to its fixed low memory footprint and is more amenable to on-chip hardware accelerators such as FPGAs.

Streaming Data Sketching on FPGA. A recent work [27] has suggested a streaming-based data transformation to adaptively learn a dictionary matrix as a subsample of incoming data measurements. Their approach is pass-efficient in a sense that they require processing each newly added data only once. The proposed framework keeps adding independent data samples to the pertinent dictionary matrix without replacing those data samples that are not frequently in use over the course of time. As such, once the dictionary matrix becomes full-rank it remains unchanged regardless of new structures that might appear in the incoming data, making the system inattentive to the changing geometry of the data due to the varying backgrounds and ambient illuminations.

Hardware-Accelerated OMP Realization. OMP is a key computational methodology to address compressing sensing and sparse approximation problems. A number of OMP implementations on CPUs [9], GPUs [28–30], ASICs [31], and FPGAs [32–34] have been reported in the literature to accelerate this computationally expensive task. The prior work on FPGA, however, have been mainly optimized for scenarios with a *static* pre-defined dictionary in which a limited number of OMP iterations (e.g., up to 32) suffices to perform the underlying data reconstruction task [32, 33, 35]. As such, these designs are not well-suited for dynamic streaming applications (e.g., background subtraction) to handle varying content of the input signals.

Object Detection Using FPGA. Object detection algorithms can be classified into three categories: (i) background subtraction [36], (ii) supervised learning [37], and (iii) point detection schemes [38]. The existing background subtraction implementations on FPGA mainly rely on static dictionary matrix to represent the background content ,e.g., [4, 5]. To the best of our knowledge, RISE is the first automated framework that enables real-time background subtraction and object detection in dynamic uncontrolled settings with varying background content.

9 CONCLUSION

This paper presents RISE, an automated framework for real-time background subtraction on SoC platforms which suits intelligent video surveillance applications. RISE takes the stream of a surveillance video as its input and learns/updates a fixed-size dictionary matrix as a subsample of the input pixels. The dictionary samples are carefully selected to capture the varying background pixels in the input video. Our framework uses the greedy OMP routine to adaptively transform the original video frames to a new set of frames in which the background pixels are eliminated based on the learned dictionary. RISE and its accompanying API provides designers with a user-friendly interface that can be used for rapid prototyping and deployment of different video surveillance applications using FPGA. Our evaluations demonstrate up to 87-fold higher throughput per energy unit achieved by RISE compared to a highly-optimized software solution on an embedded general purpose processor.

10 ACKNOWLEDGMENTS

This work was supported in parts by the Office of Naval Research grant (ONR N00014-17-1-2500) and National Science Foundation TrustHub grant (1649423).

REFERENCES

[1] V. Cevher, A. Sankaranarayanan, M. F. Duarte, D. Reddy, R. G. Baraniuk, and R. Chellappa, "Compressive sensing for background subtraction," pp. 155–168, 2008.
[2] D. Schreiber and M. Rauter, "GPU-based non-parametric background subtraction for a practical surveillance system," pp. 870–877, 2009.
[3] F. Porikli, "Achieving real-time object detection and tracking under extreme conditions," *Journal of Real-Time Image Processing*, vol. 1, no. 1, pp. 33–40, 2006.
[4] J. Oliveira, A. Printes, R. Freire, E. Melcher, and I. S. Silva, "FPGA architecture for static background subtraction in real time," pp. 26–31, 2006.
[5] C. Sanchez-Ferreira, J. Mori, and C. Llanos, "Background subtraction algorithm for moving object detection in FPGA," pp. 1–6, 2012.
[6] C. Lu, J. Shi, and J. Jia, "Online robust dictionary learning," pp. 415–422, 2013.
[7] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE signal processing magazine*, vol. 25, no. 2, pp. 21–30, 2008.
[8] J. Tropp, A. C. Gilbert *et al.*, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.
[9] A. Mirhoseini, E. Dyer, E. Songhori, R. Baraniuk, and F. Koushanfar, "Rankmap: A platform-aware framework for distributed learning from dense datasets," *arXiv preprint arXiv:1503.08169*, 2015.
[10] A. Mirhoseini, B. D. Rouhani, E. M. Songhori, and F. Koushanfar, "Perform-ml: Performance optimized machine learning by platform and content aware customization," p. 20, 2016.
[11] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern recognition*, vol. 33, no. 2, pp. 225–236, 2000.
[12] M. Karkooti, J. R. Cavallaro, and C. Dick, "FPGA implementation of matrix inversion using qrd-rls algorithm," 2005.
[13] Å. Björck, "Solving linear least squares problems by gram-schmidt orthogonalization," *BIT Numerical Mathematics*, vol. 7, no. 1, pp. 1–21, 1967.
[14] W. Hoffmann, "Iterative algorithms for gram-schmidt orthogonalization," *Computing*, vol. 41, no. 4, pp. 335–348, 1989.
[15] XILLYBUS, "http://xillybus.com/," 2017.
[16] XPower, 2012. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug440.pdf
[17] https://github.com/Bitadr/RISE, "Rise source codes."
[18] Y. Wen, Y. Lu, J. Yan, Z. Zhou, K. M. Von Deneen, and P. Shi, "An algorithm for license plate recognition applied to intelligent transportation system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 830–845, 2011.
[19] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, V. Loumos, and E. Kayafas, "A license plate-recognition algorithm for intelligent transportation system applications," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 7, no. 3, pp. 377–392, 2006.
[20] C. Arth, H. Bischof, and C. Leistner, "Tricam-an embedded platform for remote traffic surveillance," pp. 125–125, 2006.
[21] C.-N. E. Anagnostopoulos, I. E. Anagnostopoulos, I. D. Psoroulas, V. Loumos, and E. Kayafas, "License plate recognition from still images and video sequences: A survey," *IEEE Transactions on intelligent transportation systems*, vol. 9, no. 3, pp. 377–391, 2008.
[22] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image processing*, vol. 15, no. 12, pp. 3736–3745, 2006.
[23] S. D. A. LightField, 2014. [Online]. Available: http://lightfield.stanford.edu/
[24] H. R. S. D. Salina, 2014. [Online]. Available: http://www.ehu.es/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes
[25] http://www.medialab.ntua.gr/research/LPRdatabase/Still_images/difficult_cases/, "LPR database," 2017.
[26] http://ipvm.com/reports/, "Video surveillance frame rate," 2016.
[27] B. Rouhani, E. Songhori, A. Mirhoseini, and F. Koushanfar, "Ssketch: An automated framework for streaming sketch-based analysis of big data on FPGA," *23rd International Symposium on Field-Programmable Custom Computing Machines conference (FCCM)*, 2015.
[28] M. Andrecut, "Fast GPU implementation of sparse signal recovery from random projections," *arXiv preprint arXiv:0809.1833*, 2008.

[29] J. D. Blanchard and J. Tanner, "GPU accelerated greedy algorithms for compressed sensing," *Mathematical Programming Computation*, vol. 5, no. 3, pp. 267–304, 2013.
[30] Y. Fang, L. Chen, J. Wu, and B. Huang, "GPU implementation of orthogonal matching pursuit for compressive sensing," pp. 1044–1047, 2011.
[31] P. Maechler, P. Greisen, N. Felber, and A. Burg, "Matching pursuit: Evaluation and implementatio for LTE channel estimation," pp. 589–592, 2010.
[32] A. Septimus and R. Steinberg, "Compressive sampling hardware reconstruction," pp. 3316–3319, 2010.
[33] L. Bai, P. Maechler, M. Muehlberghuber, and H. Kaeslin, "High-speed compressed sensing reconstruction on FPGA using OMP and AMP," pp. 53–56, 2012.
[34] J. L. Stanislaus and T. Mohsenin, "Low-complexity FPGA implementation of compressive sensing reconstruction," pp. 671–675, 2013.
[35] A. M. Kulkarni, H. Homayoun, and T. Mohsenin, "A parallel and reconfigurable architecture for efficient OMP compressive sensing reconstruction," pp. 299–304, 2014.
[36] T. Hosaka, T. Kobayashi, and N. Otsu, "Object detection using background subtraction and foreground motion estimation," *IPSJ Transactions on Computer Vision and Applications*, vol. 3, pp. 9–20, 2011.
[37] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *International Journal of Computer Vision*, vol. 63, no. 2, pp. 153–161, 2005.
[38] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.