9-24-2019

Hyperspectral Video Processing on Resource-Constrained Platforms

Honglei Li
University of Maryland

Lei Pan
University of Maryland

Eung Joo Lee
University of Maryland

Zhu Li
University of Missouri-Kansas City

Matthew J. Hoffman
Rochester Institute of Technology

See next page for additional authors

Recommended Citation

H. Li et al., "Hyperspectral Video Processing on Resource-Constrained Platforms," 2019 10th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS), Amsterdam, Netherlands, 2019, pp. 1-5, doi: 10.1109/WHISPERS.2019.8921138.

Hyperspectral Video Processing on Resource-Constrained Platforms

Honglei Li*, Lei Pan*, Eung Joo Lee*, Zhu Li†,
Matthew J. Hoffman‡,Anthony Vodacek‡, Shuvra S. Bhattacharyya*
*University of Maryland, ECE Department and UMIACS, College Park, MD 20742, USA
†University of Missouri-Kansas City, CSEE Department, Kansas City, MO 64110, USA
‡Rochester Institute of Technology, CIS Department, Rochester, NY, 14623, USA

***Abstract*—Hyperspectral imaging offers valuable spectral diversity for scene analysis and information extraction. However, exploiting this spectral diversity involves significant challenges in performing efficient video processing, especially in resource-constrained environments. These challenges arise due to the high memory and computational requirements for hyperspectral video processing applications. This paper presents system design methods using band subset selection to address this problem. These methods are applied to develop an adaptive video processing system targeted to an Android platform. The system dynamically adapts the selected bands to process based on constraints on real-time performance and video analysis accuracy. Experimental results provide quantitative insight into trade-offs between accuracy and real-time performance under stringent resource constraints. The results also validate the effectiveness of the proposed system in performing adaptive, resource-constrained hyperspectral video processing.**

I. Introduction

Hyperspectral video processing systems (HVPSs) offer advanced capabilities for scene analytics and knowledge extraction due to their high levels of spectral diversity and spectral resolution compared to conventional video technologies. With the advancement of video acquisition techniques, HVPSs are playing increasingly important roles in video processing applications. Hyperspectral video streams provide high spectral diversity due to their high density of sampling rate in the wavelength dimension, and their capacity to incorporate diverse regions of the spectrum. Major application areas for hyperspectral image and video processing include remote sensing [9], vehicle tracking [11], and medical diagnostics [4]. However, the high density of bands involved in HVPSs brings challenges in exploiting the potential of hyperspectral imaging technology. These challenges are especially severe in the context of resource-constrained, embedded deployment, where limited memory and computational resources are available due to constraints on size, weight, power or cost.

In this paper, we develop new system design methods to address these challenges, thereby contributing novel capabilities for deploying HVPS technology in a wider variety of applications. The design methods include strategic selection of band subsets to reduce processing requirements without major loss in video analysis accuracy. Applying these design methods, we design and implement a prototype HVPS on an Android platform, and conduct experiments using a relevant hyperspectral video dataset. The experimental results demonstrate the capability of the proposed system to provide optimized hyperspectral video processing operation subject to stringent resource constraints, and to efficiently trade off real-time performance and video analysis accuracy.

II. Related Work

In recent years, advances in hyperspectral sensor technology have helped to increase the availability of hyperspectral imaging systems, which results in an increasing variety of applications for hyperspectral image and video processing (e.g., see [3]). Generally, hyperspectral imaging systems can involve hundreds, thousands or even more bands for the same scene. In addition to being more numerous, the bands employed in hyperspectral imaging systems have narrower bandwidths, thereby offering greater spectral resolution. Along with this increased resolution, however, comes the increased potential for redundancy across different bands. Thus, a natural mechanism for reducing processing requirements (e.g., to improve real-time performance or energy efficiency) in an HVPS is to select a proper subset of the available bands that provides sufficient accuracy and minimizes the storage and processing of spectral information that is redundant or is otherwise not of high relevance for the required video analysis tasks.

Various methods have been reported in the literature that are relevant to extraction of useful information from the diverse channels provided by hyperspectral imaging sensors. For example, Liu et al. provide a comparative study of different multiresolution algorithms for image fusion [8]. Wei et al. present an image fusion method for multispectral and hyperspectral images. Their method leads to less spectral error and spectral distortion compared to related fusion techniques [12]. Lin et al. compare four state-of-the-art methods for fusion of hyperspectral images [7]. Chen et al. demonstrate a pan-sharpening approach for fusing low-spatial-resolution hyperspectral images and high-spatial-resolution multispectral images of the same scene [2].

In our previous work, we demonstrated a multispectral video processing system for dynamically reconfigurable band-subset selection [6]. The system optimizes trade-offs between video analysis accuracy and processing speed. This paper goes beyond the previous work in its focus on the more challenging requirements of hyperspectral video processing,

and its targeting of highly resource-constrained devices that enable less costly, more widespread deployment.

Uzkent et al. have developed a framework for controlling hyperspectral data collection [11]. They also introduced a publicly available hyperspectral video dataset for vehicle tracking. Sobral et al. propose a stochastic tensor decomposition algorithm for robust background subtraction. Sobral's results show that red-green-blue (RGB) features are not sufficient to handle color saturation, illumination variations and problems due to shadows, while incorporating six visible spectral bands together with one near-infra-red band helps to address these limitations [10].

The distinguishing aspects of this paper include its emphasis on jointly optimizing accuracy and real-time performance in HVPSs under stringent resource constraints, with specific use of an Android smartphone platform to demonstrate the proposed methods. The paper also introduces a novel adaptive video processing system that exploits the flexibility of band-subset selection to efficiently handle time-varying requirements on the frame rate and video analysis accuracy.

A preliminary version of this work was presented in an extended abstract [5]. This paper goes beyond [5] in its integration of capabilities for adaptive band-subset selection, and its application of efficient multicore processing for enhanced real-time performance.

III. Methods

In this section, we introduce an efficient real-time HVPS that is targeted to an Android platform, and we present the underlying system design methods, which are centered on band-subset selection. The system implements background subtraction as a concrete video analysis application. The background subtraction (back-end) component of the system can readily be replaced or augmented with other video analysis techniques. This capability allows system designers to utilize in different ways the framework's capabilities for adaptive, resource-constrained hyperspectral video processing.

An important feature of the proposed HVPS is its efficiency and configurability for processing streams of hyperspectral image inputs. The proposed HVPS maintains a priority list of spectral bands that is determined through an off-line training process. The priority list is created based on each band's contribution to the overall accuracy when the bands are equally weighted. At run-time, the HVPS accesses the priority list to select N_b bands that have the highest priority, where N_b is determined based on the current real-time constraint and a constraint on video analysis accuracy. These constraints are assumed to be system parameters that can be changed dynamically. The real-time constraint specifies the minimum number of frames per second (fps) at which the system is expected to process its hyperspectral input stream.

Figure 1 illustrates the dataflow for a small-scale example configuration of the sequential, fixed-configuration (non-adaptive) first-version HVPS from [5], which we used as a starting point in this work. The dataflow graph shown in Figure 1 corresponds to a configuration in which five bands are selected for the enclosing background subtraction application, while all other available bands are ignored.

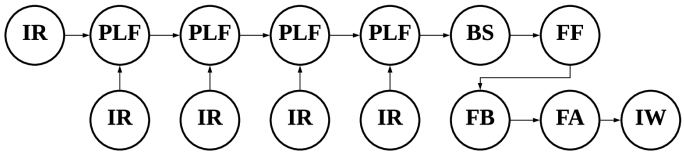

Fig. 1: Dataflow graph for an example configuration of the first-version HVPS.

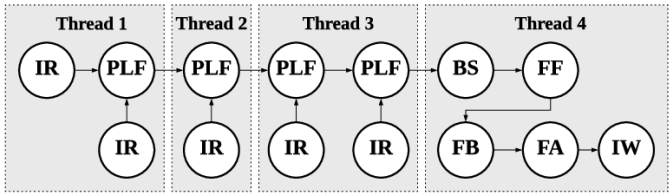

Fig. 2: Multithreaded version of Fig. 1 for mapping onto the targeted Android platform.

Each circle in Figure 1 represents an actor (signal processing module) in the dataflow graph. Brief descriptions of the actors are as follows: IR — Image Read, PLF — Pixel-Level Fusion, BS — Background Subtraction, FF — Foreground Filter, FB — Foreground Binarization, FA — Foreground Accuracy computation (for measurement and diagnostic purposes), and IW — Image Write. In each iteration of the dataflow graph, the IR actor reads from a set of files the selected bands of the next input image, and injects the image into the dataflow graph for processing.

Figure 2 illustrates the dataflow for a multithreaded version of Figure 1, which provides improved processing efficiency on the targeted multicore Android platform. This version allows more bands to be processed under a given real-time constraint, thereby improving background subtraction accuracy. The dataflow graph is composed of two parts, which we refer to as the pixel-level fusion (PLF) section (Threads 1–3) and background subtraction (BS) section (Thread 4). These sections are denoted, respectively, as S_p and S_b.

The PLF section performs pixel-level fusion to integrate pixel values from different spectral bands into a single image. The BS section then uses a Gaussian Mixture Model to perform background subtraction on the fused image. We pipeline the PLF section, using a simple pipeline of three stages, where each thread corresponds to a single stage. The three stages apply different steps of the PLF section concurrently across three successive frames of the input video stream, thereby helping to improve the achievable frame rate. The BS stage operates as an additional (fourth) pipeline stage, which processes the most recent image frame that has passed through

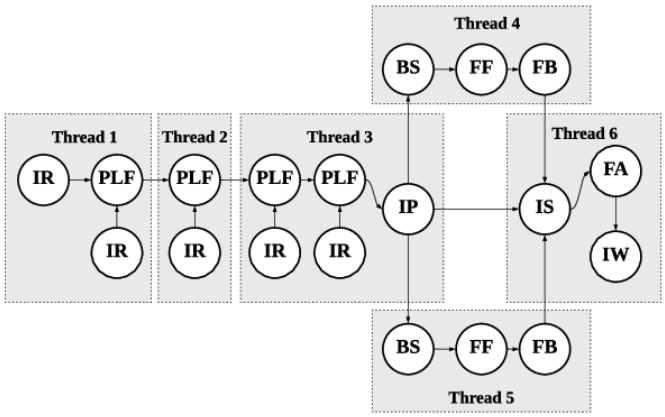

Fig. 3: Multithreaded version of Fig. 1 for $P = 6$, and $Q_p = Q_b = 3$.

all stages of the PLF section.

The pipelining process used in this design method is generalized naturally to handle arbitrary numbers of threads (e.g., for systems in which smaller or larger numbers of processing cores are available), arbitrary numbers of spectral bands, and the possibility of adding multithreading to the BS section. In the generalized form, let P, Q_p and Q_b respectively denote the total number of available threads, the number of threads allocated to the PLF section S_p, and the number of threads allocated to the BS section S_b. We assume that the available threads are utilized fully in the system design so that $P = Q_p + Q_b$. The decomposition of the available P threads into Q_p and Q_b is performed through experimentation — e.g., a binary search on Q_p can be used to arrive at a decomposition through a low-complexity experimentation process. More systematic approaches to performing this decomposition represent an interesting direction for future work.

Now if Q_p evenly divides N_b, then the number of bands allocated to each thread in S_p is simply N_b/Q_p. Otherwise, each S_p thread is assigned either $\mathit{flr}(N_b/Q_p)$ or $\mathit{clg}(N_b/Q_p)$ bands with the assignment performed in such a way that the sum of the band-to-thread assignments across S_p equals N_b. Here, flr and clg represent the floor and ceiling functions, respectively. This simple approach to distributing the processing of bands helps to keep the load of the pipeline stages balanced, which is important for throughput optimization. Here, we have assumed that $P < N_b$. The approach can be adapted easily to accommodate the case in which $P \geq N_b$; we omit the details for brevity.

Figure 3 illustrates a multithreaded design of the proposed HVPS for $P = 6$, and $Q_p = Q_b = 3$. The multithreaded configuration of the BS section incorporates two additional actors, denoted IP (Image Partitioning) and IS (image stitching). These actors, respectively, partition an image into subframes for processing across multiple threads, and integrate the different results of subframe processing into a single result.

Another important aspect of the proposed HVPS is the capability to dynamically adapt band-subset selection based on changes in real-time processing requirements or requirements in the level of video analysis accuracy (e.g., based on switching between high- and low-criticality modes of operation). Algorithm 1 gives a pseudocode sketch of the algorithm used for top-level configuration control and processing in the proposed HVPS. The while-loop ("infinite loop") in the algorithm simply indicates continuous operation that keeps processing input until the system is terminated through some sort of external/asynchronous control, such as a power-down operation.

Algorithm 1 HVPS-Toplevel

`parameter` C_r: frame rate (throughput) constraint.
`parameter` f_M: accuracy constraint.
`parameter` T: configuration monitoring interval.
`parameter` P: number of available threads.

1: **procedure** HVPS-TOPLEVEL(C_r, f_M, T, P)
2: **while** *true* **do**
3: **if** changed(f_M) **then**
4: $N_{\mathrm{b1}} := \mathit{lookup1}(T_1, f_M)$
5: **if** changed(C_r) **then**
6: $N_{\mathrm{b2}} := \mathit{lookup2}(T_2, C_r)$
7: $N_b := \mathit{max}(N_{\mathrm{b1}}, N_{\mathrm{b2}})$
8: $\mathit{process\_frames}(N_b, T, P)$

The $\mathit{changed}(p)$ function returns a Boolean value indicating whether or not the dynamic parameter p has changed (by some process external to the procedure) since system initialization (the first time the *changed* function is called on a given parameter) or since the previous call to the *changed* function (for all subsequent calls).

The video analysis accuracy metric used in the proposed HVPS can be defined based on the associated back-end video analysis functionality that is employed. For our background-subtraction-based HVPS prototype, we use the $F_{measure}$ metric, which is a commonly-used metric for assessing results of background subtraction (e.g., see [1]). We use f_m throughout the remainder of this paper as a shorthand for $F_{measure}$.

The algorithm utilizes two lookup tables, denoted T_1 and T_2. The table T_1 tabulates for different values of the accuracy metric (f_m) an estimate of the minimum numbers of spectral bands (band-subset size) that are required to achieve the specified accuracy levels. The table T_2, on the other hand, tabulates for different throughput (fps) levels, estimates on the maximum values of N_b that can be utilized without having performance fall below the throughput levels. The estimates stored in T_1 and T_2 are determined off-line through experimentation and stored in a sorted form for fast retrieval of the information at run-time. The function $\mathit{lookup1}(x)$ shown in Algorithm 1 returns the smallest value of N_b from lookup table T_1 that can achieve the specified accuracy level x. Similarly, the function $\mathit{lookup2}(x)$ returns the largest value of N_b from lookup table T_2 that can achieve the throughput level specified

by x.

The two table-lookups described above result in two candidate values for N_b, which are denoted, respectively by N_{b1} and N_{b2}. Algorithm HVPS-Toplevel then sets N_b by taking the maximum of these two candidate values, which effectively gives priority to the accuracy criterion. By changing this maximum operation to a different function, the designer can change the way the two criteria are considered in the HVPS configuration process (e.g., by prioritizing the throughput metric or applying a weighted combination to achieve a composite priority function).

After determining N_b, Algorithm HVPS-Toplevel calls *process_frames*, which encapsulates the core processing functionality (dataflow graph) of the HVPS. The function is called by passing the total number P of threads, and the band-subset size N_b that should used for the processing. The function is also called with a parameter T, which specifies the number of frames for which processing should continue before control is returned to the top-level control/configuration process represented by Algorithm HVPS-Toplevel. The parameter T effectively specifies the periodicity with which the system configuration is re-examined for a possible change in system constraints (C_r or f_M) and subsequent adaptation of processing parameters in response to such a change.

IV. Experiments

We use an Oppo N3 Android phone as the testing platform for our proposed HVPS. Oppo N3 features a Qualcomm MSM8974AA Snapdragon 801 Quad-core ARM CPU with a maximum frequency of 2.3 GHz, 2GB of RAM, and 32GB internal storage capacity. The Android OS version is 4.4.4 and Linux kernel version is 3.4.0. The hyperspectral dataset we use is generated by the DIRSIG model [11]. The dataset has 110 frames of video, where each frame has 61 spectral bands. More details on the dataset can be found in [5], [11].

For the first experiment, we collect average accuracy and frame rate results delivered by the proposed HVPS. The results are presented in terms of f_m across different values of N_b, and different multithreading configurations for each value of N_b. The data is collected for $N_b \in Z_b$, where $Z_b = \{10, 20, \ldots, 60\}$, and summarized in Figure 4. Each bar in Figure 4 represents the frame rate for a specific multithreaded or sequential configuration (Q_p, Q_b), denoted in the form "$Q_p + Q_b$", and for a specific value of $N_b \in Z_b$ (the bars for each $N_b \in Z_b$ are grouped together in the figure). Each of the six dark-shaded diamonds shows the accuracy for a given value of N_b based on the vertical-axis scale provided on the right side of the figure. The fps values displayed in Figure 4 are averaged over 50 executions for each (N_b, Q_b, Q_p) combination, and the accuracy values are averaged over all five sets of 50 executions for each N_b setting.

The figure shows that the accuracy increases monotonically with increasing $N_b \in Z_b$, and quantifies this trend of increasing accuracy. For a given (Q_p, Q_b) configuration, we see that throughput also decreases monotonically with increasing N_b. However, for a given N_b, there is no general monotonic trend

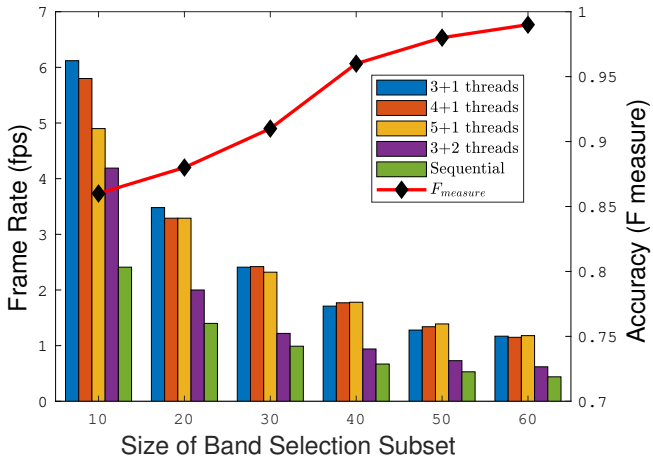

Fig. 4: Frame rate and accuracy for different N_b and different multithreading configurations.

of throughput in terms of the number $P = Q_p + Q_b$ of allocated threads. We expect that this is due to nonlinear effects related to thread allocation, such as overhead due to contention and communication across threads.

To provide more detailed insight on video processing throughput in our Android-based HVPS, a tabulation of the experimental results for the frame rate (fps) is shown in Table I. As mentioned previously, each configuration is executed 50 times in our experiments; each row of the table shows key statistics across the 50 executions associated with a given configuration. In particular, the table shows the maximum, minimum, mean, median, and standard deviation σ of measured fps for each of the configurations represented in Figure 4. From the results in Table I, we see that beyond the throughput trends discussed above in relation to Figure 4, the data in Table I demonstrates that variations in the frame rate are typically small for a given configuration (e.g., with relatively low deviation between the minimum and maximum measured values), leading to production of hyperspectral video analysis results at a fairly consistent rate.

In general, for a given value of N_b, all multithreaded versions achieved better frame rates compared to the corresponding sequential versions ($P = 1$). However, multithreading in the BS section resulted a performance degradation. This is observed when comparing the results for $Q_p = 3, Q_b = 1$ to the corresponding results for $Q_p = 3, Q_b = 2$. We anticipate that this is because the calculation for the Gaussian Mixture Model used in the BS actor, which is the core computation of the BS section, is not a bottleneck of the HVPS.

While the achieved frame rate levels, as reported in Figure 4 and Table I, are relatively low, they are sufficient for applications of resource-constrained sensing where the scene changes slowly or response time is not critical — for example, scenarios at the network edge in which the resource constrained system is used as a first-level of analysis, and is to be followed by more communication- or resource-intensive

TABLE I: Statistics on the measured frame rates for different operational configurations. The unit for each entry in the table is frames per second (fps).

Configuration	max.	min.	mean	median	σ
N_b = 10, 3+1 threads	6.30	5.92	6.12	6.12	9.42×10^{-2}
N_b = 10, 4+1 threads	5.95	5.52	6.80	5.80	8.71×10^{-2}
N_b = 10, 5+1 threads	5.02	4.80	4.90	4.89	5.38×10^{-2}
N_b = 10, 3+2 threads	4.65	4.04	4.19	4.13	1.84×10^{-1}
N_b = 10, Sequential	2.44	2.34	2.42	2.42	3.14×10^{-2}
N_b = 20, 3+1 threads	3.59	3.29	3.48	3.48	3.18×10^{-2}
N_b = 20, 4+1 threads	3.39	3.09	3.29	3.31	7.55×10^{-2}
N_b = 20, 5+1 threads	3.36	3.06	3.21	3.21	6.94×10^{-2}
N_b = 20, 3+2 threads	2.04	1.98	2.00	1.99	1.77×10^{-2}
N_b = 20, Sequential	1.41	1.40	1.40	1.41	0.91×10^{-2}
N_b = 30, 3+1 threads	2.43	2.39	2.41	2.41	1.23×10^{-2}
N_b = 30, 4+1 threads	2.46	2.35	2.42	2.43	3.37×10^{-2}
N_b = 30, 5+1 threads	2.41	2.25	2.32	2.32	5.32×10^{-2}
N_b = 30, 3+2 threads	1.24	1.21	1.22	1.23	7.10×10^{-3}
N_b = 30, Sequential	1.00	0.97	0.99	0.99	7.90×10^{-3}
N_b = 40, 3+1 threads	1.73	1.69	1.71	1.72	1.61×10^{-2}
N_b = 40, 4+1 threads	1.80	1.72	1.77	1.78	2.34×10^{-2}
N_b = 40, 5+1 threads	1.82	1.73	1.78	1.78	2.47×10^{-2}
N_b = 40, 3+2 threads	0.96	0.93	0.94	0.94	6.60×10^{-3}
N_b = 40, Sequential	0.69	0.66	0.67	0.67	9.60×10^{-3}
N_b = 50, 3+1 threads	1.30	1.23	1.28	1.29	2.02×10^{-2}
N_b = 50, 4+1 threads	1.35	1.33	1.34	1.34	6.60×10^{-3}
N_b = 50, 5+1 threads	1.41	1.37	1.39	1.39	1.46×10^{-2}
N_b = 50, 3+2 threads	0.74	0.72	0.73	0.73	4.59×10^{-3}
N_b = 50, Sequential	0.55	0.53	0.53	0.53	6.19×10^{-3}
N_b = 60, 3+1 threads	1.18	1.16	1.17	1.17	7.93×10^{-3}
N_b = 60, 4+1 threads	1.17	1.13	1.15	1.16	1.06×10^{-2}
N_b = 60, 5+1 threads	1.19	1.17	1.18	1.18	7.90×10^{-3}
N_b = 60, 3+2 threads	0.63	0.62	0.62	0.62	4.28×10^{-3}
N_b = 60, Sequential	0.45	0.44	0.44	0.44	4.06×10^{-3}

analysis at a base station if certain types of events are detected.

V. Conclusion

In this paper, we have developed new system design methods for deploying hyperspectral video processing systems (HVPSs) on highly resource-constrained platforms. Using these design methods, we have prototyped an HVPS for background subtraction on an Android platform, and conducted experiments using the prototype. The experimental results validate capabilities in the proposed HVPS framework to enable efficient design space exploration for hyperspectral video processing on resource-constrained platforms. The supported exploration demonstrated in these experiments involves complex factors, including band-subset selection, and multithreading configurations, and their impact on trade-offs between video analysis accuracy and achievable frame rate. Useful directions for future work include incorporating more sophisticated video analysis techniques in the proposed framework, and investigating design optimizations to further improve trade-offs between accuracy and throughput.

References

[1] Y. Benezeth, D. Sidibé, and J. B. Thomas. Background subtraction with multispectral video sequences. In *Proceedings of the Workshop on Non-classical Cameras, Camera Networks and Omnidirectional Vision*, 2014.

[2] Z. Chen, H. Pu, B. Wang, and G.-M. Jiang. Fusion of hyperspectral and multispectral images: A novel framework based on generalization of pan-sharpening methods. *IEEE Geoscience and Remote Sensing Letters*, 11(8):1418–1422, 2014.

[3] L.-J. Ferrato and K. W. Forsythe. Comparing hyperspectral and multispectral imagery for land classification of the lower Don River, Toronto. *Journal of Geography and Geology*, 5(1):92–107, 2013.

[4] J. Freeman, F. Downs, L. Marcucci, E. N. Lewis, B. Blume, and J. Rish. Multispectral and hyperspectral imaging: applications for medical and surgical diagnostics. In *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 700–701, 1997.

[5] H. Li, L. Pan, Z. Li, M. J. Hoffman, A. Vodacek, and S. S. Bhattacharyya. Design methods for hyperspectral video processing on resource-constrained platforms. In *Proceedings of the Hyperspectral Imaging & Applications Conference*, Coventry, UK, October 2018. 2 pages in online proceedings.

[6] H. Li, K. Sudusinghe, Y. Liu, J. Yoon, M. van der Schaar, E. Blasch, and S. S. Bhattacharyya. Dynamic, data-driven processing of multispectral video streams. *IEEE Aerospace & Electronic Systems Magazine*, 32(7):50–57, 2017.

[7] H. Lin and J. Chen. Comparison of several hyperspectral image fusion methods for superresolution. In *Proceedings of the International Conference on Image, Vision and Computing*, 2018.

[8] Z. Liu, E. Blasch, Z. Xue, J. Zhao, R. Laganiere, and W. Wu. Objective assessment of multiresolution image fusion algorithms for context enhancement in night vision: A comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):94–109, 2012.

[9] S. B. Serpico and L. Bruzzone. A new search algorithm for feature selection in hyperspectral remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 39(7):1360–1367, 2001.

[10] A. Sobral, S. Javed, S. Ki Jung, T. Bouwmans, and E. Zahzah. Online stochastic tensor decomposition for background subtraction in multispectral video sequences. In *Proceedings of the International Conference on Computer Vision Workshop*, pages 946–953, 2015.

[11] B. Uzkent, M. J. Hoffman, and A. Vodacek. Integrating hyperspectral likelihoods in a multidimensional assignment algorithm for aerial vehicle tracking. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(9):4325–4333, 2016.

[12] Q. Wei, J. Bioucas-Dias, N. Dobigeon, and J.-Y. Tourneret. Hyperspectral and multispectral image fusion based on a sparse representation. *IEEE Transactions on Geoscience and Remote Sensing*, 53(7):3658–3668, 2015.