Video Anomaly Detection in Real Time on a Power-Aware Heterogeneous Platform

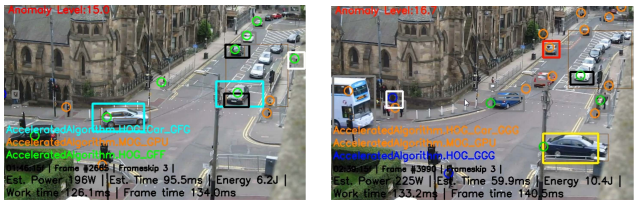
Calum G. Blair and Neil M. Robertson, *Senior Member, IEEE*

***Abstract*— Field-programmable gate arrays (FPGAs) and graphics processing units (GPUs) are often used when real-time performance in video processing is required. An accelerated processor is chosen based on task-specific priorities (power consumption, processing time, and detection accuracy), and this decision is normally made once at design time. All three of the characteristics are important, particularly in battery-powered systems. Here, we propose a method for moving selection of processing platform from a single design-time choice to a continuous run-time one. We implement Histogram of Oriented Gradients (HOG) for cars and people and Mixture-of-Gaussians motion detectors running across FPGA, GPU, and central processing unit in a heterogeneous system. We use this to detect illegally parked vehicles in urban scenes. Power, time, and accuracy information for each detector is characterized. An anomaly measure is assigned to each detected object based on its trajectory and location compared with learned contextual movement patterns. This drives processor and implementation selection so that scenes with high behavioral anomalies are processed with faster but more power-hungry implementations, but routine or static time periods are processed with power-optimized and less accurate slower versions. Real-time performance is evaluated on video data sets including i-LIDS. Compared with power-optimized static selection, automatic dynamic implementation mapping is 10% more accurate, but draws 12-W of extra power in our testbed desktop system.**

***Index Terms*— Event detection, field-programmable gate array (FPGA), graphics processing unit (GPU), heterogeneous, object detection.**

I. Introduction

SURVEILLANCE systems have recently become more commonplace, more portable, and capable of handling more complex tasks. The process of selecting or designing a processing system for any surveillance application involves meeting some performance constraints (computation time must be fast enough for real-time performance) and optimizing others (power consumption should be minimized and algorithm accuracy should be maximized). Improving any one of these characteristics will result in tradeoffs in the others. Here, we propose a real-time video surveillance system for detecting anomalous behavior in an urban setting and analyze its performance. In this scenario, our video analysis system contains a heterogeneous set of processors and its power supply will be constrained in some way, such as relying on batteries. The ability to perform timely and accurate detections with minimized power consumption is important. At different time periods, power should be prioritized more than speed (or frame rate), and vice versa, depending on the level of any perceived threat. This allows a power-constrained real-time system to quickly react to relevant changes in its environment while conserving power in periods of inactivity.



(a) (b)

Fig. 1. Anomaly detection in an urban scene (BankSt data set). When an anomalous event (car stopping in an unusual area) is detected, the detection speed is prioritized and power consumption increases as a result. (a) Pedestrians and vehicles moving normally. (b) Car parked in a forbidden location (red square).

A. Motivation

In recent years, algorithms of increasing computational complexity have been developed, allowing more accurate detection or classification of objects of interest. In conjunction with the rise in pixel data volume from higher resolution imaging devices, this has led to requirements to process higher volumes of data in a timely fashion—ideally in real time. Many of these algorithms are embarrassingly parallel, so two technologies have been employed to allow accelerated processing of video data, for online and offline real-time tasks [1], [2]. These devices represent two different approaches to the problem of parallelization; field-programmable gate arrays (FPGAs) allow temporal parallelization by building up long pipelines of simple arithmetic operations specifically matched to the problem in question and permit spatial parallelization by duplicating these units as broadly as possible to process multiple parts of the image at once. Graphics processing units (GPUs) use thousands of existing cores optimized for fast arithmetic to massively parallelize the calculations required by an algorithm. These approaches have

Manuscript received April 2, 2015; revised July 17, 2015; accepted October 2, 2015. Date of publication October 19, 2015; date of current version October 27, 2016. This work was supported in part by Thales Optronics, in part by the Engineering and Physical Sciences Research Council under Grant EP/J015180/1, and in part by the Ministry of Defence University Defence Research Collaboration in Signal Processing. This paper was recommended by Associate Editor S. Shirani.

C. G. Blair is with Institute for Digital Communications, The University of Edinburgh, Edinburgh EH8 9YL, U.K. (e-mail: c.blair@ed.ac.uk).

N. M. Robertson is with Vision Laboratory, Heriot-Watt University, Edinburgh EH14 4AS, U.K.

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the author.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2015.2492838

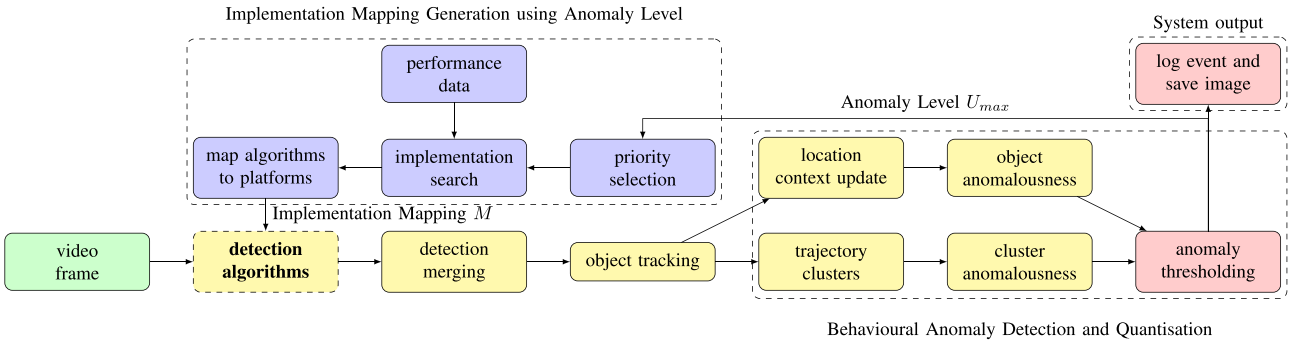


Fig. 2. Flow diagram of frame processing, anomaly detection, and algorithm mapping loop. Computationally expensive operations are in bold. Boxes or groups with dotted lines show our contributions: #1—switchable hardware platforms for detection algorithms, #2—behavioral anomaly detection, and #3—implementation mapping.

tradeoffs; FPGAs have lower power consumption, but require considerably more time and knowledge to program [3]. They also have a long history and acceptance of being deployed in military systems. GPUs are programmed using a more familiar software design flow and can allow greater numerical precision due to their native reliance on floating-point arithmetic, but draw considerably more power.

The choice of platform to deploy an algorithm onto is typically made at design time, as part of a decision about how a task should be partitioned onto hardware. Typically, once a decision to deploy to a specific platform is made, the consequences (higher power consumption, reduced detection capability, etc.) remain part of that system and must be accepted throughout its use. Delaying the choice of processing architecture until the system is deployed allows the system to trade off power consumption and frame processing time *on the fly*, *in response to changes in scene conditions*. For example, when no activity is seen, or no anomaly is present, we can conserve power at the expense of frame rate and processing time. If a higher level of anomaly is present (for example, a pedestrian entering an unusual area, or a car driving the wrong way down a road or parking in a forbidden location), any of these anomalous actions could represent a threat. For vehicle-mounted systems performing a surveillance task, this may mean a threat to its occupants, so in this case, we would ideally process data and identify anomalies more quickly; immediate power consumption is less important. These priorities dynamically change with scene content.

B. Contributions

In this paper, we describe and analyze the performance of our heterogeneous system that uses central processing unit (CPU), FPGA, and GPU to allow real-time detection of objects and anomalies. It autonomously chooses which set of processors to run an algorithm or algorithm stage on. This decision is made dynamically and depends on the level of anomaly seen in the video stream. We evaluate this on two video data sets and demonstrate detection of this behavior in both data sets while reducing the overall power consumption. Vehicles that park or stop in forbidden or unusual areas are used to represent anomalous behavior. An example is shown in Fig. 1(b), where vehicles are forbidden from parking on the double yellow lines at the roadside. An anomalous vehicle is denoted by a red box.

Our contributions are as follows.

1) We describe a hardware platform of heterogeneous processors (FPGA, GPU, and CPU), with multiple possible implementations of computationally expensive algorithms that can be run on it. We characterize each implementation's power consumption, processing time, and detection accuracy. Our implementations and their platforms can be switched between dynamically.
2) We introduce an algorithm for quantizing the level of behavioral anomaly in a video stream using object and motion detectors.
3) We use a hardware mapping algorithm that uses anomaly level to prioritize reduced power consumption or shorter processing time. This then dynamically selects algorithm implementations using their performance characteristics. This set of implementations is used to process subsequent video data.

These contributions are shown in the overall system diagram in Fig. 2.

C. Paper Layout

This paper is structured as follows. Section II places this work in context by describing the related work in the areas of anomaly detection, parallelized algorithm implementations, and power-aware computing. Section III describes our system at a high level and details the components used to perform object and anomaly detection. It also covers our mechanism for dynamically switching between algorithm implementations. Section IV describes the video data sets and evaluation procedure and presents the results. Section V discusses these in more detail, along with our system's limitations. Finally, Section VI restates our results and describes directions for future work.

II. Related Work

This paper is concerned with algorithms at multiple levels of abstraction, from feature extraction in images up to behavior analysis. We aim to process video data in real time, so here we also describe accelerated processing and how to schedule or allocate processing tasks to minimize power consumption.

A. Anomaly Detection

Loy *et al.* [4] define anomalous events or unusual behavior in video as being in one of the three categories (those very different from the training set, those which are ambiguous and rarely present in the training set, and those with only weak visual evidence). Humans can fail to detect the latter two types, especially during longer tasks. Morris and Trivedi [5] consider surveillance in urban scenes and note that as scenes become more unstructured, identifying unusual events becomes harder. Detection in urban scenes, with multiple classes of traffic participants and fewer restrictions on directions, is more difficult than behavior recognition on a highly structured motorway, particularly when vehicles in urban environments may be stationary for long time periods. Anomaly detection spans several levels of abstraction, from individual object detections that can be grouped into trajectories and then clustered into common paths in a scene [6]. This produces clear trajectories, but requires offline analysis. Piciarelli and Foresti [7] describe a method for online clustering of object locations into trajectories. A sliding temporal window allows tracks to be matched to clusters; the window expands as the track length increases. Given a learned set of tracks clustered into trajectory trees, the likely behavior of a new object can be predicted, as the most common clusters and parent–child cluster transitions are known [7]. Morris and Trivedi [8] describe a system for detecting U-turn events in traffic, but we look here at the problem of detecting parked vehicles.

Albiol *et al.* [9] use spatiotemporal maps to identify these events in the Imagery Library for Intelligent Detection Systems (i-LIDS) data set. This relies on human operators labeling regions where parking is forbidden. Regions are evaluated as belonging to the background or not, and hence distinct objects are not detected or classified. Lee *et al.* [10] also process some events from i-LIDS, using a 1D transformation to detect vehicles parking illegally, and demonstrate real-time performance on CPU.

B. Accelerated Object Detection

Work on real-time object detection has attracted much interest recently, particularly given the possibilities for detection from embedded systems fitted to vehicles. A common pedestrian detection algorithm is "histogram of oriented gradients" (HOG) [11]. HOG is split into three computationally expensive algorithm stages: 1) image resizing; 2) feature extraction; and 3) classification. Many improvements and variations to the overall algorithm are documented in [12]. Dollár *et al.* [13] have demonstrated that performing feature extraction on fewer scales and rescaling the resulting features are also effective. Current state-of-the-art detectors are still based on this work; they extract various color and shape features before classifying these using a support vector machine (SVM) or boosted classifier. Multiple hardware implementations of HOG and its derivatives have been proposed, on various devices including GPU [1], [14], FPGA [15], and a hybrid system [16], with feature extraction done on the FPGA and SVM classification handled by the GPU. A comparison of these platforms in a heterogeneous system is performed in [17], where the processing time, detection accuracy, and system power consumption of the HOG algorithm are all compared. The image resize, feature extraction, and SVM classification subtasks can be allocated to GPU, FPGA, or CPU. The algorithm runs fastest when all tasks are performed on GPU and uses the least power when CPU and FPGA are used. A mix of GPU and FPGA processing allows a reasonable tradeoff between power and frame rate.

A car detection algorithm has been built using a similar approach, using HOG with deformable part models [18].

C. Platform Selection and Power-Aware Computing

FPGAs and GPUs occupy different points in design space. Allocating tasks (those defined as stages of an algorithm or algorithms in their entirety) onto processors is a form of *design space exploration*. Cope *et al.* [19] have compared FPGA and GPU for accelerating low-level image processing operations such as convolutions and enhancement algorithms. GPU performance decreases with increasing kernel size, but there is no single platform that is always most suitable and the data flow of the desired algorithm must always be considered. Wu *et al.* [20] reinforce these conclusions while taking power into account.

Points in design space are evaluated for optimality using a Pareto curve [21]. Optimal points dominate nonoptimal ones if they improve on at least one characteristic. Eventually, a Pareto front of all the optimal points is formed. Prioritizing one characteristic over another will move between these points.

Yu and Prasanna [22] explore power-aware resource allocation for static tasks. Quinn *et al.* [23] explored tradeoffs in assigning discrete algorithm stages to FPGA or CPU for pixel processing algorithms. Tradeoffs were expressed in terms of chip area versus instruction latency, rather than accuracy and power. However, there is a limited body of work on using power consumption information to map tasks to processors in video. The main work we are aware of in this field is by Llamocca and Pattichis [24]. They perform dynamic multiobjective optimization in a power–performance–accuracy space (in this context, performance refers to processing time). This is done for pixel processing algorithms, including gamma correction and contrast enhancement, where the design space is populated by implementations with different clock frequencies and bit widths. A user-chosen accuracy level drives dynamic selection, and the selected implementation is loaded via FPGA dynamic reconfiguration.

Blair and Robertson [25] considered power-aware dynamic allocation on algorithms operating at a higher level of abstraction, where multiple heterogeneous accelerators are used. We expand on this work here by adding another video evaluation, further algorithm and implementation details, and additional experimental results, including time taken to switch between selected implementation mappings.

III. System Details

Our system detects objects and motion in a video view of a scene, uses contextual information obtained from previous behavior to assign an anomaly level to scene activity, and uses this to select the optimal algorithm implementations to

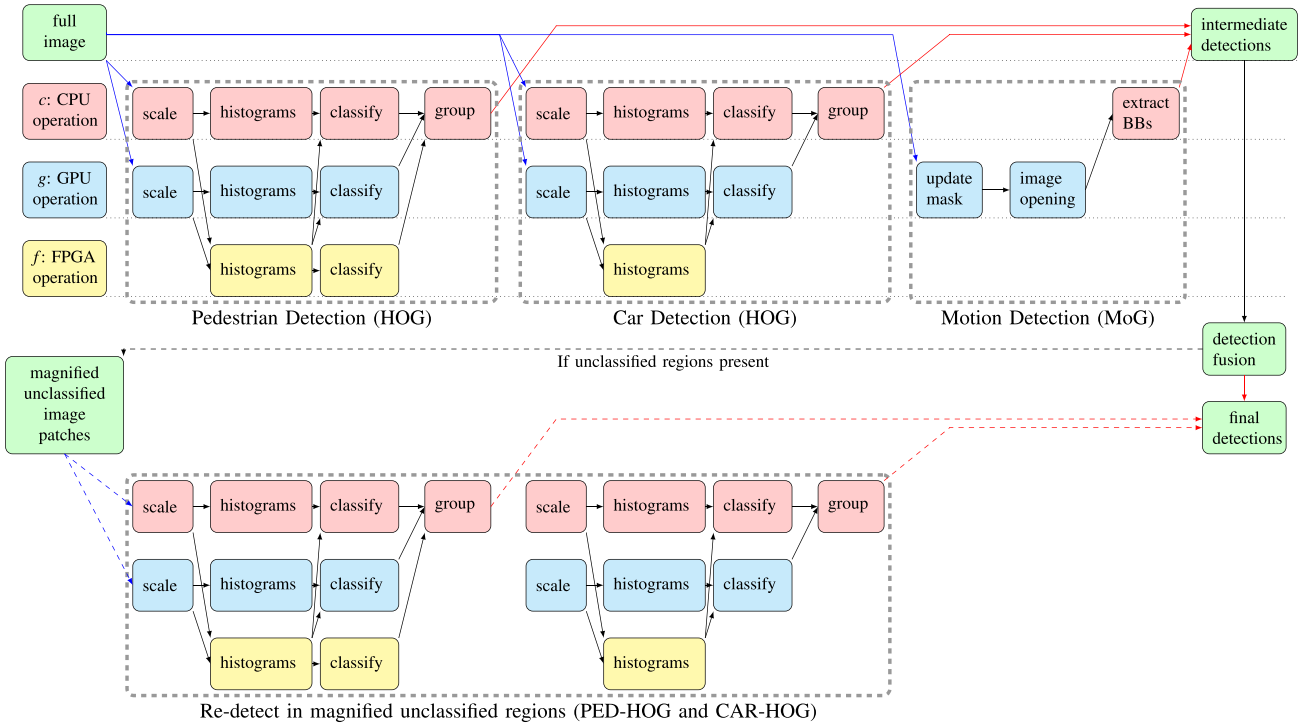


Fig. 3. Hardware implementations of computationally expensive detection algorithms showing all possible mappings. Mnemonics for implementation stages are referred to in the text; HOG with image resizing on GPU, histogram extraction on FPGA, and classification on GPU is labeled gfg. The bottom half shows attempted reclassification of regions that are detected as containing only motion. This passes data through the same detectors as above.

process the next incoming frame. We detect both pedestrians and cars; this allows identification of common traffic participants and, hence, their behaviors within the environment. The system processes offline videos, but operates at 25 frames/s and dynamically calculates the number of frames to drop to maintain this real-time processing rate. The time to decode the input video and the time to display or annotate an output image are not counted within this window (as we assume that live video would be captured without needing to be decoded, and the only outputs required are infrequent snapshots of anomalous events as they occur).

Fig. 2 gives a high-level overview of our system. Where relevant, we present performance information on aspects of the system in this section; the overall results for the anomaly detection task are presented in Section IV.

In Fig. 2, the detection algorithms block processes each image and generates bounding boxes describing object location and classification. These are used by the later algorithm stages. The detectors run every frame and do not take temporal information into account. This is instead evaluated at a higher level by the tracker, which matches new detections to existing tracks. Computationally expensive implementations of car, pedestrian, and motion detectors and algorithms run here. An expanded view of this stage is given in Fig. 3. These algorithms each had at least one accelerated version available; these are listed in Table I.

A. Hardware Platform

The platform used contained a host CPU (2.4-GHz dual-core Intel Xeon), an FPGA (Xilinx ML605 board with an XC6VLX240 device), and GPU (NVidia GeForce 560Ti). Data could be transferred via direct memory access (DMA) between each processor over PCI express (PCIe), as shown in Fig. 4. In the FPGA, the DMA and PCIe transfer logic ran at 250 MHz and the image processing application was clocked at 160 MHz. The resource use is given in Table II. Table II shows the total FPGA resources used to implement the PED-HOG and CAR-HOG detectors and the interface logic.

TABLE I
ALGORITHMS AND IMPLEMENTATIONS USED

Algorithm (* means compute-intensive)	Implementation(s)		
Pedestrian Detection*: PED-HOG (§III-B)	FPGA	GPU	CPU
Car Detection*: CAR-HOG (§III-C)	FPGA	GPU	CPU
Motion Detection*: MoG (§III-D)		GPU	
Tracking: Kalman Filter (§III-E)			CPU
Trajectory Clustering (§III-F)			CPU
Bayesian Motion Context (§III-G)			CPU

TABLE II
TOTAL RESOURCE UTILIZATION ON Xilinx XC6VLX240 FPGA. THIS INCLUDES PCIe AND DMA LOGIC AND BOTH DETECTORS: THE HISTOGRAM EXTRACTION STEP FOR CAR-HOG AND THE HISTOGRAM AND CLASSIFICATION STEPS FOR PED-HOG

Resource	Resource Use	Percentage
Slice Registers	81888 of 301440	27%
Slice LUTs	77623 of 150720	51%
BlockRAM	217 of 832	26%
DSP48 Multiplier	108 of 768	14%

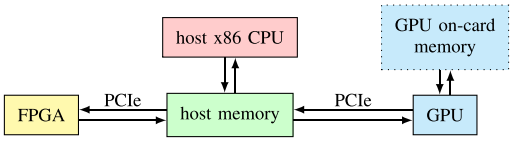


Fig. 4. Desktop PC system with heterogeneous processors. Each processor is connected over PCI express and can access the host's main memory. The FPGA and GPU each have private on-card memory.

TABLE III

FRAME PROCESSING TIME t_{proc} FOR FRAME n USING THE HOG IMPLEMENTATION LISTED, WHERE THE PREVIOUS FRAME $(n-1)$ WAS PROCESSED WITH THE SAME (COLUMN 2) OR DIFFERENT (COLUMN 3) IMPLEMENTATION. COLUMN 4 SHOWS TIMES FOR DIFFERENT t_{proc} AS A PERCENTAGE OF THE SAME t_{proc}. THE IMAGE SIZE IS 1024×768. IMPLEMENTATION ACRONYMS ARE DEFINED IN SECTION III-B

Impl.	t_{proc} (ms) for given impl. when previous impl. was:		Time for $a \to b$, as % of $a \to a$
	$a \to a$ (same)	$a \to b$ (different)	
ccc	776.6	765.1	98.5
ggg	55.5	55.7	100.3
gfg	119.1	123.1	103.3
cfc	646.4	640.1	99.0
cff	96.6	95.1	98.4
gff	85.5	88.2	103.1

Application logic for each algorithm was autogenerated from Xilinx System Generator software. There was sufficient capacity within the FPGA to hold the implementations in Fig. 3; because of this, partial dynamic reconfiguration was not required. Further platform details are given in [17].

B. Pedestrian Detection (PED-HOG)

The accelerated versions of the HOG pedestrian detector described in [11] are used. These are split into three tasks: 1) image resizing or scaling; 2) generation of gradient histograms in local cells; and 3) block histogram generation, normalization, and SVM classification. These correspond to stages in the original algorithm [11]. A note on mnemonics: *xyz* refers, in general, to scaling on platform *x*, histograms on *y*, and SVM classification on *z*. For example, an implementation that rescales the image on the GPU, generates histograms on the FPGA, and classifies on the GPU is given the gfg mnemonic. Similarly, ccc means an implementation where scaling, histograms, and classification are done on the host CPU. This is illustrated in Fig. 3, where the arrows on the left show the paths that can be taken through the algorithm. As Table III shows, the different implementations could be dynamically switched between frames with no loss of performance. Each version was trained on the INRIA pedestrians data set [11]. Measurements of power, processing time, and accuracy for each implementation when tested on the test portion of this data set are given in Table IV. The detection accuracy for each version is shown in Fig. 5 and is comparable with the original algorithm.

TABLE IV

PERFORMANCE DETAILS FOR ALGORITHM IMPLEMENTATIONS ON 770×578 VIDEO. THE PROCESSING TIME (ms), OVERALL SYSTEM POWER CONSUMPTION (W), AND DETECTION ACCURACY [LOG-AVERAGE MISS RATE (%)] ARE SHOWN. THE BASELINE POWER CONSUMPTION WAS 147 W

Algorithm	Platforms	Time (ms)	Power (W)	Miss rate (%)
PED-HOG	*ggg*	17.6	229	52
	cff	23.0	190	62
	gff	27.5	186	61
	gfg	39.0	200	59
	cfc	117.3	187	59
	ccc	282.0	191	53
CAR-HOG	*ggg*	34.3	229	89
	cfc	175.6	189	94
	gfg	60.0	200	92
	ccc	318.0	194	89
MoG	GPU	8.1	202	N/A

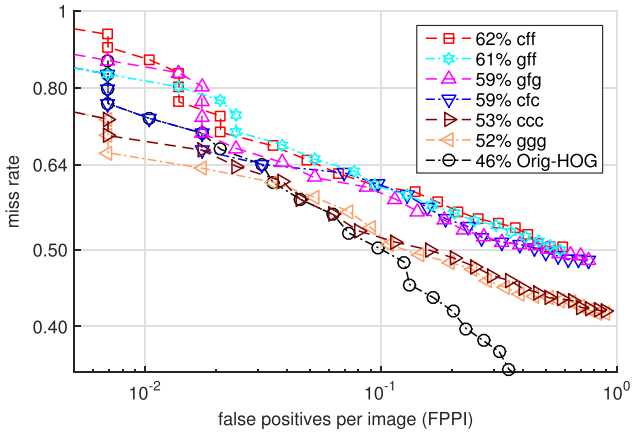
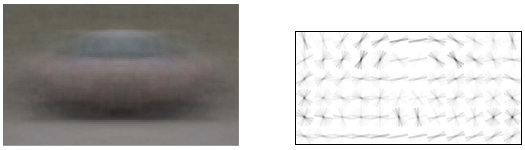


Fig. 5. Detection error tradeoff curve for PED-HOG detector, showing false positives per image against miss rate.

While existing accelerated implementations of HOG are available, they do not take partitioning into account (i.e., do all the work on GPU) or use static partitioning [16]. In addition to writing new implementations where necessary, our modification efforts for existing platforms (CPU and GPU) were focused on profiling (to identify the most appropriate stages to partition an algorithm and transfer intermediate data between processors) and interoperability, i.e., ensuring that cell histograms generated on FPGA could be transferred and classified on GPU and that this produced acceptable results.

C. Car Detection (CAR-HOG)

Given that we had existing real-time implementations of HOG running on multiple architectures, these were modified to detect cars. The CPU and GPU versions of the HOG OpenCV code were modified using the parameters given in [26] for vehicle detection. Similarly, the cfc and gfg implementations from [17] were modified to detect cars. In all cases, detection was performed at multiple scales in real time. While this results in a reduction in accuracy when compared with more



(a) (b)

Fig. 6. Training the CAR-HOG algorithm. (a) Composite image of all training samples. (b) Final oriented-gradient SVM car model.

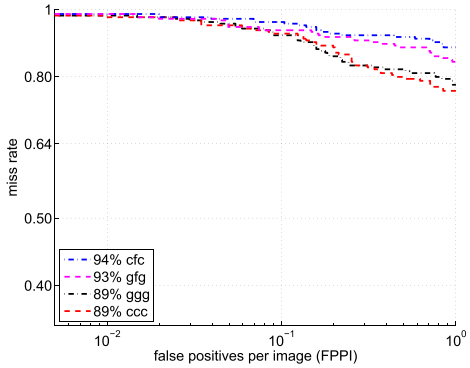


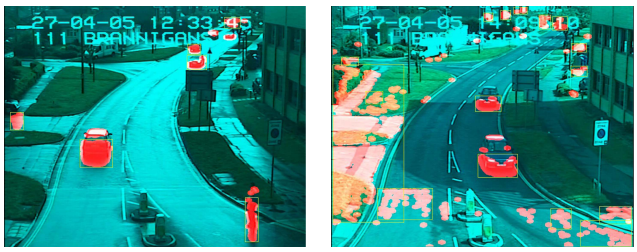
Fig. 7. Detection error tradeoff curve showing false positives per image (FPPI) for CAR-HOG.

sophisticated detectors, the implementation time required to produce multiple implementations of a different algorithm would have been prohibitive.

The car detector was trained using images from the 2012 Pattern Analysis, Statistical Modelling and Computational Learning (PASCAL) visual object classes challenge [27]. Image windows containing cars were extracted and resized so that each car was 48 pixels high. Cars were selected if they were not marked as occluded, truncated, or difficult in the ground truth and were originally $\geq$40 pixels high. A total of 212 images were used, made up of original and horizontally flipped copies of 106 cars. The SVM model was trained as described in [11]. Fig. 6(a) shows a composite model of all positive training samples, and Fig. 6(b) shows the learned oriented-gradient model. Strong gradients on the vertical pillars are visible. Fig. 7 shows detector accuracy as a false positives per image (FPPI) curve. This is less accurate than the pedestrian version; we speculate that this is due to the reduced set of training images and the wide variations in appearance of cars being viewed head-on and side-on. During testing, vans and lorries were also capable of being detected.

D. Motion Detection (MoG)

The Mixture-of-Gaussians (MoG) algorithm was used to perform background subtraction, thus segmenting foreground objects [28]. There were several motivations behind this; this technique allowed detection and then classification of small or distant objects, below the minimum size of the HOG detectors, and also detection of moving objects close together. The alternative to this would have been running both HOGs on a magnified version of the entire image, which would have not been possible with real-time processing. The OpenCV GPU implementation was used [29].



(a) (b)

Fig. 8. MoG motion detection algorithm on GPU. (a) Motion bounding boxes successfully identified. (b) False positives due to illumination changes and camera shake.

This was then morphologically opened (as both these steps were computationally expensive) before transferring to the host, where contour detection was performed and bounding boxes were produced. Every foreground region that was not yet classified was passed to additional pedestrian and vehicle detectors (as shown in the bottom half of Fig. 3), so early identification of overlaps led to reduced processing at a later stage. An overlap criterion was used to compare bounding boxes; for pairs with $\geq$90% intersection, the smaller one was removed. Bounding boxes are defined as $B = \{x_1 \ldots x_2, y_1 \ldots y_2\}$ and $\text{area}(B) = (x_2 - x_1) \times (y_2 - y_1)$. Thus, B_i and B_j are compared and B_i is discarded if

$$\frac{B_i \cap B_j}{\text{area}(B_j)} \geq 0.9 \;\&\; \text{area}(B_i) < \text{area}(B_j)\,. \tag{1}$$

This performed well, as shown in Fig. 8(a). Automatic gain correction or shake from the camera would occasionally incorrectly detect motion, as shown in Fig. 8(b). In this case, all motion detection for that frame was discarded.

E. Detection Fusion and Object Tracking

The system had two sources of object detections: classified bounding boxes generated directly by CAR-HOG or PED-HOG, or unclassified regions of interest generated by the motion detector. Regions from the latter algorithm were extracted and magnified, and then passed to each HOG algorithm again. This allowed most objects to be classified as human or vehicle, rather than simply an object in motion. As we expect humans and vehicles to move at different speeds and frequent different regions in the scene, this allowed more accurate contextual scene information to be gathered than if we had simply detected moving objects without attempting to classify them.

To normalize inter-object distances and speeds, all detections were projected via an affine transform onto a base plane and then smoothed with a constant-velocity Kalman filter, as shown in Fig. 9. This also matched per-frame detections to persistent trackers for each object; an elliptical distance measure was used to select a tracker to match a detection to, where the long axis of the ellipse pointed in the direction of travel of the tracker. When a tracked object was stationary, this became the Euclidean distance measure. This approach allowed unclassified detections to be matched to existing classified tracks. From this algorithm stage onward, the volume

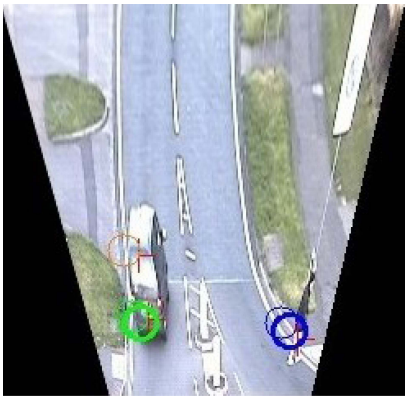


Fig. 9. Transformed base plane image showing the bottom part of the i-LIDS scene, including, car (green circle) and pedestrian (blue circle) object trackers.

of data to be processed was low enough that acceleration was no longer required, and all further processing was done on the host CPU.

F. Clustering Trajectories

To cluster tracked points into trees of trajectories, we reimplement a clustering algorithm described by Piciarelli and Foresti [7], who used it for detecting anomalies in traffic flow. The original work looked at a motorway junction where fast moving vehicles can take a small number of routes through the scene, but the scenes we apply this to are less structured. We consider urban scenes with two types of traffic participants, multiple entrances and exits, and areas where objects stop moving for long periods and may be considered part of the background. Our objective is to assign a trajectory T to one of a set of clusters $\mathbf{C} = \{C_1, C_2, \ldots, C_n\}$. Each trajectory $T_i = (t_0, t_1, \ldots, t_n)$ represents tracked detections over several frames, where each $t_i = (x_i, y_i)$. Each cluster $C_k = (c_0, c_1, \ldots, c_n)$ is a vector of points $c_j = (x_j, y_j, \sigma_j^2)$ with location and variance. Clusters are arranged in a tree structure and have zero or more children. A tree (starting with a root cluster) describes a single point of entry to a scene from one of the edges, and all observed paths taken through it from that point. Given a new (unmatched) trajectory T_u, all root clusters and their children are searched to a given depth. T_u is assigned to the closest C_k if the distance D is below a threshold. This is done via

$$D(T_u, C_i) = \min \left[\frac{d(t_i, c_j)}{\sqrt{\sigma_j^2}} \right], \quad j \in \{\lfloor (1-\delta)i \rfloor \ldots \lceil (1+\delta)i \rceil\} \tag{2}$$

where $d(t_i, c_j)$ is the Euclidean distance between t_i in T_u and c_j in C_k, and j defines the range in C_k to search over. The lower and upper search bounds of the cluster points in C_k are governed by $\delta = 0.4$. Thus, when matching long trajectories to longer clusters, more possible matches are allowed. This takes account of subsequent objects in one cluster not moving in exactly the same manner. Once a point is matched to a cluster, the corresponding cluster element c_j is updated by t_i with a learning factor $\alpha = 0.05$.

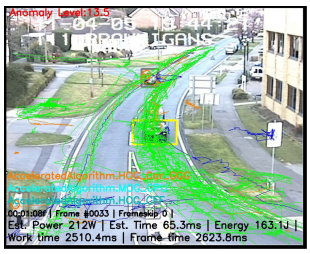
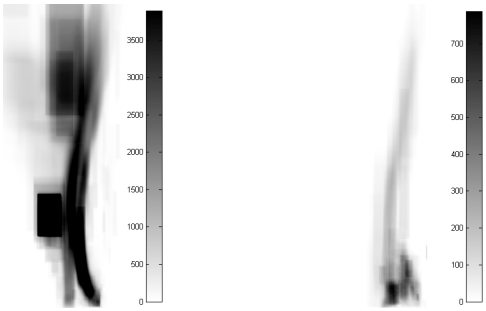


Fig. 10. Object tracks grouped into trajectory clusters and retransformed onto the camera plane. Green, blue, and orange tracks show cars, pedestrians, and undetermined (motion-only) objects, respectively.



(a) (b)

Fig. 11. Presence intensity maps for (a) cars and (b) pedestrians in the PV3 data set. The color bar shows the average occupation rate of that pixel, and the x and y axes relate to the base plane projection of the scene (partly shown in Fig. 9).

If no matching clusters are found, a new root cluster is created. As trajectories matched to a cluster are updated with new points, the points can be updated, the cluster can be extended, or new child clusters split off as points begin to diverge. Fig. 10 shows this approach on real data; several clusters representing pedestrian and vehicle motion can be seen.

G. Contextual Location Data

Contextual knowledge makes use of known information about the normal or common actions of participants within a scene. Simple features such as position and motion data can capture various anomalous behaviors including vehicles parked in an unusual location or those moving the wrong way down a road. Here we use unsupervised learning to learn scene context. We use the classified bounding boxes from the object and motion detectors during training sequences to build up 2D per-pixel base plane heatmaps of object position or presence; these are shown in Fig. 11.

In a similar manner, Fig. 12 shows motion maps for pedestrians and vehicles in x and y. Velocity data were obtained from object trackers observed during the learning clips. For an object moving at $\mathbf{v} = (v_x, v_y)$ and occupying $(x_1, \ldots, x_2, y_1, \ldots, y_2)$, the map $\bar{\mathbf{v}}$ representing average per-pixel velocity is updated

$$\bar{\mathbf{v}}_{(x_1,\ldots,x_2,y_1,\ldots,y_2)} = (1-\alpha)\bar{\mathbf{v}}_{(x_1,\ldots,x_2,y_1,\ldots,y_2)} + \alpha \mathbf{v} \tag{3}$$

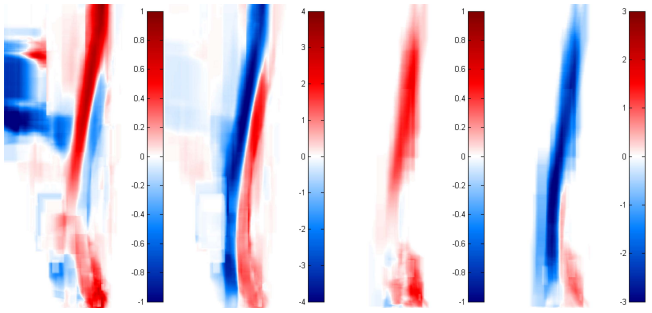


Fig. 12. Base plane motion maps built using learned movement from different object classes in PV3. (b) On-road vertical motion away from (blue region) and toward the camera (red region) is distinct and clearly defined. The color bar denotes the average velocity at that pixel in pixels/frame. The x and y axes correspond to the base plane image of the scene from Fig. 9. (a) Car v_x. (b) Car v_y. (c) Ped v_x. (d) Ped v_y.

where $\alpha = 0.0002$. Fig. 12 shows motion maps for pedestrians and vehicles in x and y. This approach works well for most conceivable traffic actions. However, this does not capture more complex interactions between traffic participants. As this is unsupervised learning, errors from the object detectors will propagate to the heatmaps.

H. Detecting Anomalies

Using the algorithm in Sections III-F and III-G, we can define an anomalous object as one that is present in an unexpected area or present in an expected area but moves in an unexpected manner. It follows that these objects will relate to events that are not present or underrepresented in the training data and not representative of normal traffic flow. We use a Bayesian approach to determine if an object's velocity in the x and y directions should be marked as anomalous, based on the average velocity $\bar{v}$.

First, we define an object anomaly measure U_O as the probability of a pixel being associated with an anomaly $p(A|D)$, given a detection at that pixel

$$U_O = p(A|D) = \frac{p(D|A)p(A)}{p(D|A)p(A) + p(D|\bar{A})p(\bar{A})} \tag{4}$$

where $p(A)$ and $p(D|A)$ are constants (given that the probability of detecting an anomaly at any point within the image is a constant and the likelihood of detecting an object at any anomalous pixel is also constant). We set $p(\bar{A}) = 1 - p(A)$, and we can, however, vary $p(D|\bar{A})$, which is based on the similarity between the observed motion $\mathbf{v}$ and the learned mean per-pixel motion $\bar{\mathbf{v}}$, and is in the range (0.01, 0.99). First, we find d_v, a distance measure between $\bar{\mathbf{v}}$ and $\mathbf{v}$. This is separately done for v_x and v_y

$$\begin{aligned} &\text{if } \operatorname{sign}(\bar{v}) == \operatorname{sign}(v) \ \&\ |v| > |\bar{v}| \\ &\quad d_v = \operatorname{sign}(\bar{v}) \times \max(W \times |\bar{v}|, |\bar{v}| + W) \\ &\text{otherwise, } \ d_v = \operatorname{sign}(\bar{v}) \times \min(-0.5 \times |\bar{v}|, |\bar{v}| - w) \end{aligned}$$

where W and w are forward and reverse directional constants. We perform linear regression, obtaining a line gradient

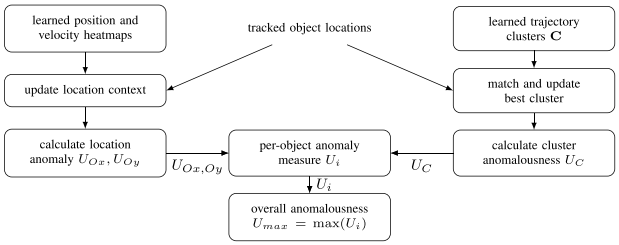


Fig. 13. Flow diagram of contribution #2: anomaly detection process. Using tracked object locations identified by the detection algorithms, learned cluster, position, and velocity information is used to assign anomaly levels to individual objects. Finally, an overall per-frame behavior anomaly level is quantified.

of $a = (0.01 - 0.99)/(d_v - \bar{v})$. We then obtain an intermediate value k, which is substituted into $p(D|\bar{A})$

$$k = av + b \tag{5}$$

$$p(D|\bar{A}) = \max(0.01, \min(0.99, k)) \tag{6}$$

and b is calculated in a manner similar to a. Finally, (4) is used to obtain U_{Ox}. This is repeated for U_{Oy}.

We combine this with U_C, an anomaly measure describing information about the abnormality of the current cluster associated with an object. U_C is based on one of the two measures: transits and transitions. When an object moves from one cluster to any of its children or leaves the field of view, the number of *transits* through that cluster is incremented. For a parent cluster C_p with children C_{c1} and C_{c2}, the number of trajectory *transitions* between the parent node and each of its children is also recorded. This builds up a frequency distribution between all children of any cluster. Anomalous trajectories can thus be identified. These metrics are combined; if C_i is a root node

$$U_C(C_i) = \frac{1}{1 + \text{transits}(C_i)} \tag{7}$$

otherwise if C_i is one of the n child nodes of C_p

$$U_C(C_i) = 1 - \frac{\text{transitions}(C_p \to C_i)}{\sum_{j=1}^{n} \text{transitions}(C_p \to C_j)}. \tag{8}$$

The anomaly measure U_i for object i with an age of τ frames can then be given

$$U_i = w_o \frac{\Sigma_1^{\tau_i} U_{Ox}}{\tau_i} + w_o \frac{\Sigma_1^{\tau_i} U_{Oy}}{\tau_i} + w_c U_C(C_i). \tag{9}$$

Weights w_o and w_c are set such that two out of three anomaly indicators are needed to flag an object as anomalous. U_{Ox} and U_{Oy} are running totals averaged over τ. For every frame, the overall anomaly measure is then $U_{\max} = \max(U_i)$. This process is shown in Fig. 13. Taken together, objects that stop in areas where the normal behavior is to move at speed in a particular direction are detected by both the object anomaly and the cluster trajectory algorithms (as a stationary object will still advance in time out of the frequently transited parent cluster and ultimately into a child cluster of its own). If an object is flagged as anomalous for a minimum time threshold t_A, its location and classification are logged and a snapshot of the video frame is saved.

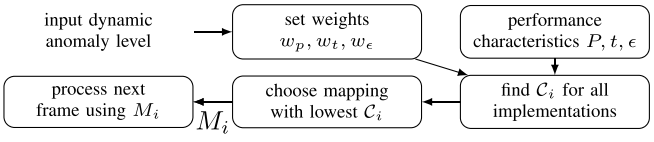


Fig. 14. Flow diagram of contribution #3: hardware mapping process. Using a dynamic anomaly level as input, the lowest cost mapping is found using stored performance characteristics and then used to process the next frame.

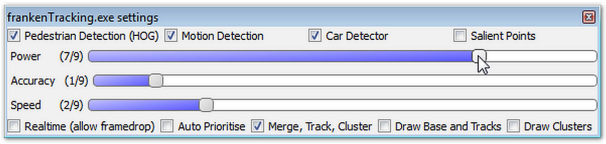


Fig. 15. User- or algorithm-driven priority selection. Moving a slider to the right represents increased weight given to that performance characteristic. Other sliders are automatically moved to the left to compensate. See the supplementary material for videos of this process.

I. Dynamic Algorithm Mapping

By this stage, we have reduced a video frame to $U_{\max}$, a single scalar describing the anomaly level in that frame. We now use this to choose a mapping or set of algorithm implementations M to process the next frame with. This process is shown in Fig. 14. M is recalculated every time a frame is processed, so new implementations can be selected in response to changing scene characteristics. Any mapping M_i must include one implementation of PED-HOG, CAR-HOG, and MoG (m_{ped}, m_{car}, and m_{mog}) and can be any combination of the paths shown in Fig. 3. If a frame takes longer to process than real time, subsequent frames are skipped to maintain real-time performance. Time spent processing every part of the algorithm in Fig. 2 is counted and used when calculating the number of frames that should be skipped. This includes time spent attempting to reclassify small unclassified regions as shown in the bottom half of Fig. 2.

We assume that a higher level of anomaly in the scene should be responded to with increased processing resources to allow inference in a more timely fashion and at the expense of power consumption. Time periods with low or zero anomaly levels will cause power to be reduced, resulting in longer processing times, lower accuracy, and more frames being dropped. We define $R = \{w_p, w_t, w_\epsilon\}$ as the current prioritization setting, with each w being individual prioritizations for time, power, and miss rate. Ten credits are allocated between the three weights; the use of integer weights was a limitation of the user interface library. R is set by assigning a given fraction of credits to each weight. This can be done manually (by always assigning the maximum weight possible to speed or power; see Fig. 15 for an example) or automatically, by maximizing speed when $U_{\max}$ is above a threshold or power when below. (Some hysteresis is built in using two separate thresholds.)

Using the calculated priorities and set of desired algorithms, implementation mapping is then run. M is selected by choosing the lowest cost implementation. For a given mapping M_i, a cost function is used as

$$\mathcal{C}_i = w_p P_i + w_t t_i + w_\epsilon \epsilon_i. \tag{10}$$

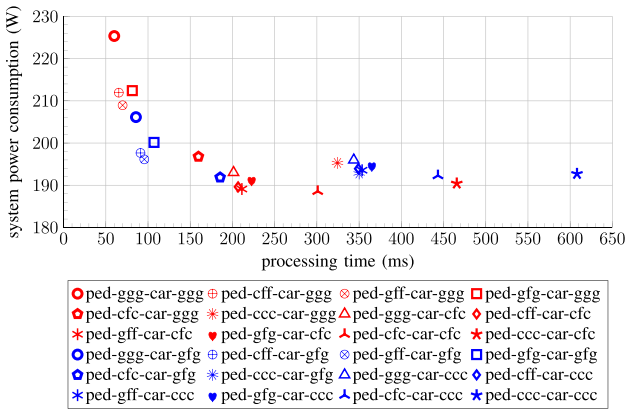


Fig. 16. Design space exploration of power consumption versus processing time for every car, pedestrian, and motion detector implementation. All points include motion detection on GPU.

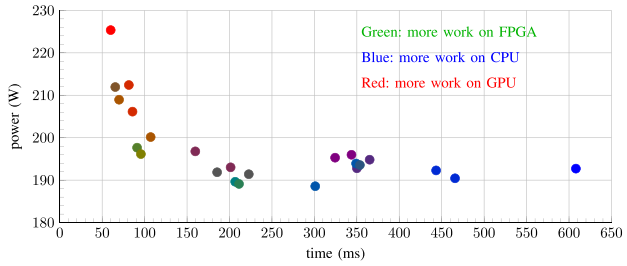
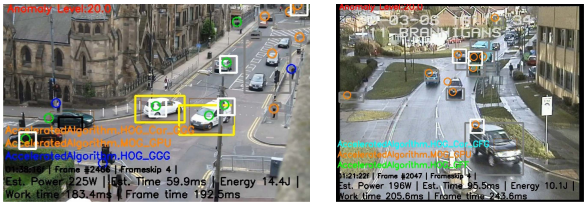


Fig. 17. Power and time plots of all possible solutions, where each solution consists of one CAR-HOG, PED-HOG, and MoG detector, as described in IV. A greener dot represents a solution with most operations mapped to FPGA, while the bluer and redder dots represent those with most operations mapped to CPU or GPU, respectively.

Here, the implementation performance characteristics P, t, and ϵ represent system power consumption, frame processing time, and detection accuracy expressed as miss rate. These costs are incurred while processing a frame, that is, running $m_{\text{ped}}, m_{\text{car}}$, and m_{mog} in M_i. These are calculated from the values shown in Table IV. t_i is the sum of the processing time of M_i. P_i is the average power consumed while processing $m_{\text{ped}}, m_{\text{car}}$, and m_{mog} [i.e., $(P_{\text{car}}t_{\text{car}} + P_{\text{ped}}t_{\text{ped}} + P_{\text{mog}}t_{\text{mog}})/t_i$]. ϵ_i is a ranked measure of the miss rate of m_{car} and m_{ped}. An M with ped-ggg, car-ggg, mog-gpu would have $t_i = 60$ ms, $P_i = 225$ W, and ϵ_i as a ranked accuracy measure. The points in Fig. 16 show discrete t_i and P_i calculations for every possible mapping.

Normalization of the P, t, and ϵ values in Table III is not required. The calculated values can be used directly, as any decision on normalization is ultimately application specific and represents the relative priority designers or users would assign to longevity of operation of any system versus fast detection capability. The normalization approach used (including the use of integer weights for R) balances all of these factors and ensures that multiple mappings are used where appropriate.

This tradeoff is graphically shown in Fig. 17, where points in stronger red, green, or blue signify the majority of algorithm stages being mapped to FPGA, GPU, or CPU, respectively. The lowest cost mapping is then selected and used to process the next frame and generate new detections.



(a) (b)

Fig. 18. Examples of video quality and effect on detection ability. In (a) BankSt, classification of most objects as either pedestrian (blue circles) or car (green circles) is possible. In (b) i-LIDS, many tracks remain as unclassified moving objects (orange circles).

IV. Methodology and Results

Having described the system and the parameters of interest (namely, R and its effect on M), we now test its ability to detect parked vehicles on two video data sets of scenarios. These are described here, followed by our methodology.

A. Scenarios

We use two representative scenarios of anomalous behavior in an urban environment, both gathered from a static camera. The *BankSt* scenario involves a static camera monitoring a busy four-way crossroads in daylight, at 720×400 and 30 frames/s with little camera shake present [see Fig. 1(b)]. Vehicles are often stationary for extended periods while waiting at the traffic lights in both directions. BankSt has an 18-min training and a 4-min test video. The training data contain no parked vehicles, only normal traffic patterns. The testing data contain one event where a vehicle parks in a forbidden location.

The i-LIDS data set [30] is a U.K. government video data set of various indoor and outdoor scenarios, used for the evaluation of anomaly detection algorithms. We use a parked vehicle detection scenario, PV3, for real-time evaluation. This consists of 5.5 hours each of training and testing footage of a scene as shown in Fig. 18(b). Ground truth of every event is provided as a time-stamped description. The video is in color visible light range and is 720×576 at 25 frames/s. It overlooks an urban road; several turnings are present on the left and right, and traffic often queues to enter a roundabout beneath the frame. Video clips from several months apart, in various weather conditions, and at different times of day or night are provided. Strong shadows, camera shake, camera movement in between clips, videotape artifacts, and noise interfere with reliable scene analysis. PV3 is poorer quality and considerably more challenging than our BankSt data. This is apparent in the comparison shown in Fig. 18. It is, however, instructive as an example of the data that anomaly detection algorithms may have to work within real-world tasks.

B. Methodology

The processing platform described in [17] was first evaluated to ensure that no substantial delay was present when differing implementations were selected. Frame processing times were recorded when either maintaining constant mappings or switching them between subsequent frames. Switching between different mappings involved changes on up to three platforms. FPGA implementation switching involved setting or clearing bits to select a different processing pipeline; this was done as part of the command to start a DMA transfer and involved no reconfiguration or overhead. CPU and GPU implementation switching involved taking different software branches while a frame was being processed. This is included in the processing time, so any switching costs were thus negligible. As shown in Table III, the time difference when switching mappings was always under 4% of the nonswitching time. This ensured that (10) did not need to include a switching cost.

In each test we performed, we compare the performance of three prioritization methods with the following labels.

Speed: Speed manually prioritized, auto prioritization OFF.

Power: Power manually prioritized, auto prioritization OFF.

Auto: auto prioritization ON, controlled by anomaly level.

For both data sets, we manually register points in the camera image onto a base plane for a homography transform. Unlike in [9], this is the only manual intervention required; we do not restrict the detection of anomalies to a known area in the image through masking, but allow them to occur anywhere. This also allows expansion to multiple scenarios.

The clusters and heatmaps are then learned by running a training clip containing no anomalous behavior, 17–20 min in length. The same clusters and heatmaps were used for all tests. Following Loy's approach [4], we define an anomalous event as observed behavior that is absent or rarely present in the training data. Thus, vehicles parking in forbidden areas are used as anomalous events; we define these as objects that have U_i over a set threshold for a time period t_A.

We first test our system on BankSt to demonstrate extensibility to a variety of surveillance scenarios, and then perform a complete evaluation on i-LIDS. The single event in the 4-min BankSt test clip was detected using all three prioritization modes [as shown in Fig. 1(b)], along with one false positive, in each case caused by stationary vehicles at the traffic lights. More analysis is performed on i-LIDS as it is a larger and more challenging scenario. It is also an open data set that has benchmarks and results already reported in the literature. The default i-LIDS criteria require that an anomaly event must persist for 60 s before it can be logged, and then it must be recorded within 10 s following this to count as a true positive. As our algorithm needs only 10–15 s to detect events, events are treated as true positives if the recorded start time is within 70 or 75 s after the ground-truth time recorded when the vehicle actually parks. However, we require anomalous events to be localized to the object that is parked wrongly; this is more discriminative than the original i-LIDS criteria, which require only a binary alarm signal in the presence of an anomaly. We evaluate with the anomaly detection window t_A set to 10 and 15 s, log detected events, and compare them with ground-truth event data.

C. Results

We first summarize object detection performance by considering each measurement (power, speed, and accuracy) separately. This allows us to gather data on the individual

implementation performance characteristics required for generating M. We then discuss anomaly detection on i-LIDS.

1) Object Detection Performance Characteristics: The performance for object detection is given in Table IV. Here, the pedestrian detectors are always more accurate than the car versions. The GPU-based detectors are faster and consume more power, while implementations that perform more processing on FPGA have reduced power consumption and accuracy.

There are several causes of the accuracy differences seen in the final column of Table IV; these are due to algorithm and platform differences between processors. Our FPGA implementation is most inaccurate, due to numerical and algorithm simplifications. Fixed-point arithmetic must be used on FPGA, as this results in significantly lower resource use. This causes errors in precision to accumulate when compared with single-precision floating-point versions seen on CPU and GPU. The FPGA histogram generation step omits the Gaussian weighting of pixel blocks at a slight cost in accuracy. The FPGA classification stage also uses a simplified block normalization step compared with the software implementations (L1 normalization followed by a square root, compared with two-stage L2 normalization present in the original algorithm; see [26]). Both of these decisions were taken to reduce the FPGA resource use and minimize expensive calculations such as division operations. Discrepancies between the CPU and GPU accuracy measurements are caused by the execution of floating-point operations in a different order to achieve maximum parallelization and slight differences in the pixel cell generation code. These per-platform differences, when combined, account for the variations seen in miss rate. More details are given in [17].

Power and time data were obtained while playing a 770×578 video, and the detection accuracy figures refer to the testing portion of the data set used for training (either INRIA or PASCAL). Power measurements in Table IV were gathered using a plug-in meter that recorded total system power consumption. We plot this power–time tradeoff in Figs. 16 (showing the exact placement of each mapping solution) and 17 (showing the proportion of each work carried out on a particular processor). Here the solutions form a Pareto curve, stretching from {ped-ggg, car-ggg, mog-gpu} at the top left via {ped-gff, car-gfg, mog-gpu} to {ped-cfc, car-cfc, mog-gpu} in the bottom center, with other points shadowed. If a single mapping was both lowest power and fastest, it would overshadow all other mappings and always be the optimal mapping to use in all situations; this is not the case.

Due to the design choices made when producing implementations for each platform, the ranking of accuracy measurements between implementations is fixed and does not depend on the choice of data set used to train or test the individual detectors; there will be no data set where, e.g., gff is more accurate than ggg. The cost function expresses the tradeoff of choosing a less accurate implementation *of the same algorithm evaluated on the same data set* over the one that uses more power; thus, the performance information in Table IV is not data dependent. In support of this conclusion, Dollár *et al.* showed that ranking of object detector performance remains relatively consistent between different data sets [12].

TABLE V

DETECTION OF PARKED VEHICLE EVENTS ON i-LIDS PV3 FOR EACH PRIORITIZATION MODE. TRUE POSITIVES (TPs), FALSE POSITIVES AND NEGATIVES (FPs AND FNs), PRECISION (p), AND RECALL (r) ARE SHOWN. F_1-SCORES ARE SHOWN FOR OPERATIONAL AWARENESS (OA) AND EVENT LOGGING (EL)

Priority	TP	FP	FN	p	r	$F_{1,OA}$	$F_{1,EL}$
for $t_A = 10$ seconds							
power	4	44	26	0.083	0.133	0.0957	0.1317
speed	6	51	25	0.105	0.194	0.1254	0.1913
auto	6	53	25	0.102	0.194	0.1226	0.1912
for $t_A = 10$ seconds, daylight only							
power	4	29	23	0.121	0.148	0.1294	0.1475
speed	6	40	22	0.130	0.214	0.1510	0.2118
auto	6	42	22	0.125	0.214	0.1466	0.2115
for $t_A = 15$ seconds							
power	2	15	29	0.118	0.065	0.0910	0.0652
speed	8	13	23	0.381	0.258	0.3259	0.2594
auto	4	14	26	0.222	0.133	0.1797	0.1342
for $t_A = 15$ seconds, daylight only							
power	2	10	29	0.167	0.065	0.1067	0.0652
speed	8	8	23	0.500	0.258	0.3752	0.2601
auto	4	10	26	0.286	0.133	0.2030	0.1345

2) Anomaly Detection Performance Characteristics: All detected anomalous events are logged and a snapshot is taken. We compare this with the ground truth using both precision $p = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$ and recall $r = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$ measures, where TP, FP, and FN are true positives, false positives, and false negatives, respectively. The F_1-score is also calculated

$$F_1 = (\alpha + 1)rp/(r + \alpha p) \tag{11}$$

where α is a recall bias measure, set to 0.55 for real-time operational awareness (so that false alarm rate is reduced and operator confidence is maintained) or 60 for event logging (so that all plausible events are logged), giving $F_{1,\mathrm{OA}}$ and $F_{1,\mathrm{EL}}$, respectively. The details of these are given in Table V for both t_A measures, for the two manual and one automatic prioritization modes. There are no ground-truth events in the night clips, but these generate a large proportion of false positives, so we also show the results for daylight-only clips (those labeled day or dusk). From Table V, prioritizing for speed always improves accuracy, but prioritizing reduced power consumption reduces accuracy. Automatic prioritization provides a compromise between these two extremes. Extending the t_A to 15 s allows a significant reduction in the number of false positives.

In Table VI, we show the performance details for all prioritization modes. The t_{all} column shows per-frame execution time including overheads as a percentage of the time available per video frame, $t_{\mathrm{src}} = 40$ ms. The processing time column t_{work} does not include overheads (i.e., assumes that a raw frame is already in memory and annotated per-frame output is not needed so that only events are logged). The system runs faster than real time here. When running individually, some implementation mappings are able to process every frame in real time, as shown by the processing times in Table IV.

TABLE VI

PROCESSING PERFORMANCE FOR POWER, SPEED, AND TIME PRIORITIZATIONS. FRAME SKIP PERCENTAGE ($fskip$), PROCESSING TIME WITH AND WITHOUT OVERHEADS (t_{all} AND t_{work}) COMPARED WITH SOURCE FRAME TIME (t_{src}), AND TOTAL ESTIMATED AND MEAN ESTIMATED POWER ABOVE THE BASELINE (P^*_{work}) ARE SHOWN. THE BASELINE (IDLE) POWER CONSUMPTION WAS 147 W. F_1 ACCURACY SCORES ARE ALSO SHOWN

Priority	$fskip$ (%)	t_{all}/t_{src} (%)	t_{work}/t_{src} (%)	P^*_{work} (W)	$F_{1,OA}$
power	75.8	125.4	87.9	49.1	0.0910
speed	59.5	124.3	81.7	72.8	0.3259
auto	66.5	127.8	83.4	61.9	0.1797
daylight only					
power	75.5	125.4	87.9	49.1	0.1067
speed	60.1	124.2	82.0	78.4	0.3752
auto	66.0	122.0	83.4	61.8	0.2030

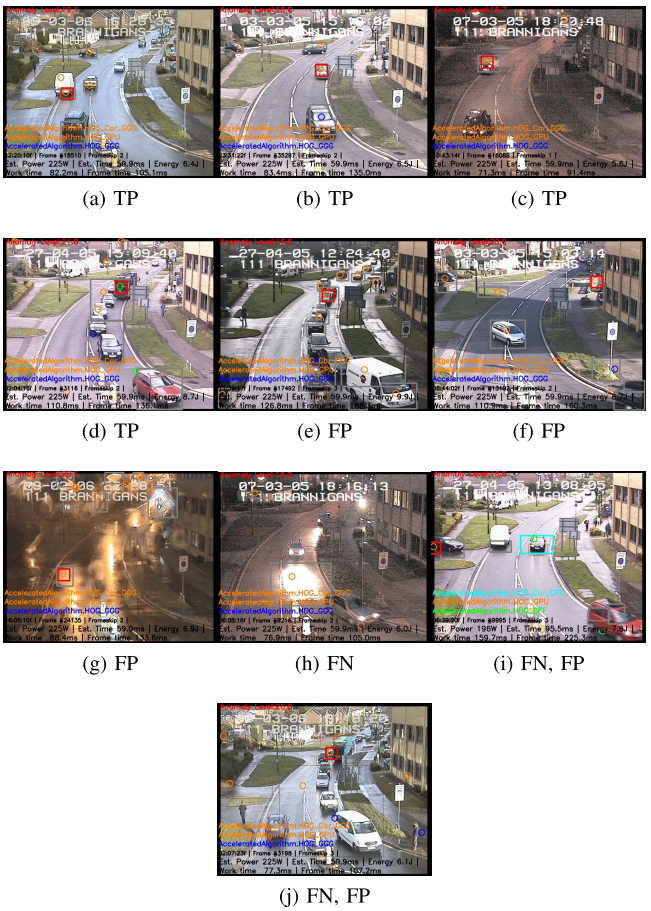
Algorithms are run sequentially, so for 40-ms frames, we can run one or two of the detection algorithms for every frame, but not all three. The overall system is, however, able to process a video stream in real time by skipping a proportion of frames.

The percentage of skipped frames is in the $fskip$ column; as expected, when we optimize for speed, then frames are processed faster, so fewer frames are dropped. From Table IV, power prioritization reduces the overall power consumption while speed increases it, showing that the prioritization modes have some effect and behave as expected. The power row in Table VI maps most processing to FPGA, so here 75% of frames are skipped. When speed is prioritized, the speed prioritization maps everything to GPU, so when all work is done on GPU, then only 59% of frames are skipped. For auto, the implementations used vary, so an average frameskip value is shown. Fast algorithms that skip fewer frames have higher accuracy, as shown by the relationship between the $fskip$ and F_1 values in Table VI.

Example detections, logged after t_A seconds, are given in Fig. 19. While true positives are detected in a variety of conditions (including at dusk), even in the best case, only up to 50% of events are detected. There are also many false positives and negatives, mainly due to shortcomings in the object or motion detectors. False positives can be caused by the MoG algorithm flagging patches of empty road as foreground [Fig. 19(g)], but mitigating this may cause slow-moving or stationary traffic in that region to be ignored. Errors can also be caused by limitations of the HOG implementations (such as in Fig. 20) or failure to detect occluded objects [Fig. 19(h)]. Video quality has an impact here too; as shown in Fig. 18, higher quality video allows more reliable classification (into pedestrian and car classes) of moving objects. Other limitations such as in Fig. 19(i) are also apparent; as we have failed to associate the anomalous event with the object causing it, this incident counts as *two* errors (FN and FP).

V. ANALYSIS

Here we concentrate on the most significant results from these experiments and compare them with existing work.



(a) TP (b) TP (c) TP (d) TP (e) FP (f) FP (g) FP (h) FN (i) FN, FP (j) FN, FP

Fig. 19. Examples of positive detections and failure modes of anomaly detector on i-LIDS. Anomalies are marked by red boxes. (a)–(d) True positives of various objects in different locations in quiet and busy surroundings. (e)–(g) False positives from slow-moving traffic, an object moving off-screen, and incorrect background subtraction, respectively. (h) Car is occluded behind the road sign on the right; this is treated as a false negative. (i) False negative and a false positive: the anomaly detector identifies the car on the left instead of the van parked beside it. (j) Anomaly identified outside the allowed time window, so counts as a false negative and false positive.

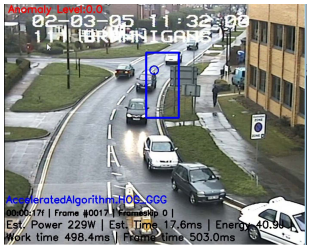


Fig. 20. Failure mode of object detection stage, showing a false positive (blue rectangle representing a pedestrian). These affected the measured accuracy, both directly and through the cluster and object abnormality measures.

Considering the tradeoffs between power consumption, accuracy, and time, we note that in this case, we are constrained by the real-time processing requirement. Power and accuracy are therefore the main characteristics we can vary, as we have limited flexibility over the time requirement.

The key results are found in Tables V and VI. From Table VI, there is a 29-W range in average power consumption above the baseline. The platform used was based on a desktop PC, and hence was not optimized for low power

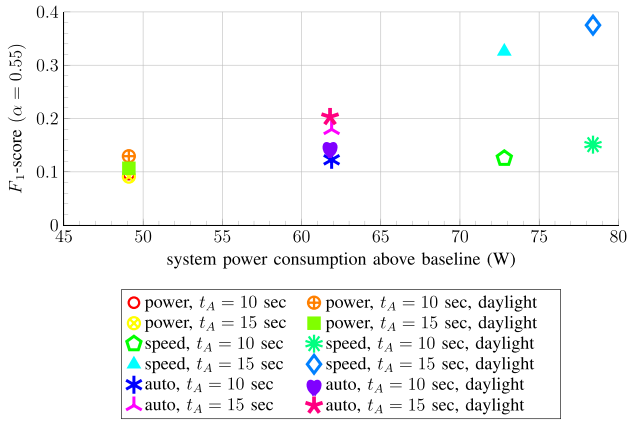


Fig. 21. Error rate versus power consumption: F_1 operational awareness scores ($\alpha = 0.55$) against power consumption for various time thresholds for various lighting conditions.

consumption (hence the relatively high 147-W idle power). With the system run with speed prioritized and the FPGA disconnected, the power consumption P_{avg} was 208 W, or 62 W above the baseline. P_{avg} consumed under auto prioritization with FPGA enabled was 61.9 W, but this is specific to this data set; video data sets with fewer moving objects or longer idle periods would have a considerably lower P_{avg} than this. This is also because of the lack of gff and cff implementations for CAR-HOG, which would reduce P_{avg} further.

We consider P_{avg} versus accuracy in Fig. 21. Using *auto* prioritization allows a 10% increase in accuracy over the *power*-optimized option, at a cost of 12 W extra in P_{avg}. A further 17% improvement in F_1-score (moving from *auto* to *speed*) costs an extra 17 W. When compared with *speed*, the 12-W power reduction moving to *auto* reduces the accuracy by 45% of the baseline, while the fully optimized *power* option loses 72% accuracy for 32% in power savings from the best case. This is most apparent for longer detection windows, on daylight-only clips with higher levels of anomalous behavior. These measurements and Fig. 21 itself show a clear relationship between power consumption and overall detection accuracy.

A. Comparison With State-of-the-Art Work

Following other researchers' definitions of anomalous events (events not present in and different to the training data) [4], we argue here that the detection of parked vehicles is a reasonable representation of the more general task of detecting anomalous events in video. The only previous work to perform parked vehicle detection on the full i-LIDS PV3 data set has been by Albiol *et al.* [9]. They used spatiotemporal maps and manually applied lane masks in each clip to signify which scene regions could have valid detections. Using this approach, they reached $p-$ and $r-$ values of 100% in some clips. Performance information is not given, but video frames are downscaled to 320×240 to decrease the evaluation time. They note that, as an approach for detecting slow-moving and stationary vehicles, background subtraction has various limitations. They also encountered similar difficulties as those shown in Fig. 18. Lee *et al.* [10] perform illegal parked vehicle detection on short clips extracted from the i-LIDS set; they achieve close-to-real-time performance at the original resolution and detect all events in the clips with no false positives. Thus, if we take into account only accuracy measurements, we are unable to improve on these existing results. Here, however, we consider a novel and different problem: that of automated power-aware anomaly detection rather than monitoring lane occlusion in a manually masked region. The only manual intervention we required in these experiments was to register points in each camera viewpoint to perform an affine transform onto the base plane. In the future, this step could be done automatically.

The detection algorithms in this paper are the most time- and power-intensive components of this system. Accuracy in these could be improved by implementing more sophisticated algorithms on one or more accelerated platforms. However, this would require considerable development time if each version were to be implemented. As shown in [12], many of the current most accurate pedestrian detectors are based on HOG, so the hardware acceleration techniques documented in this paper and the overall conclusions would apply if more accurate versions were substituted in the future. As the Pareto curve in Fig. 17 shows, any such implementation with known accuracy would always involve a compromise between power and speed too; each measurement may become a priority at any point in time.

The other work concerning power-aware resource allocation in video is by Llamocca and Pattichis [24]. This paper is concerned with tasks at higher levels of abstraction than their work, but we are able to use the calculated anomaly level to dynamically drive selection of the optimal mapping, rather than relying on a user-selected accuracy level. This results in a fully automated system. We are thus able to tie low-level performance constraints such as power or energy consumption to high-level accuracy of detected behavioral events.

VI. Conclusion

We have described a real-time system that performs parked vehicle detection along with classification of humans and cars. This can select between various algorithm implementations on a mixture of FPGA, GPU, and CPU, each of which has different power consumption, implementation runtime, and algorithm accuracy characteristics. These are dynamically selected based on a feedback loop driven by the number of objects in a video perceived to be behaving anomalously (in this case, parking in forbidden areas). This allows us to dynamically trade off power consumption against detection accuracy and shows benefits when compared with fixed power- or speed-optimized versions. We evaluated this on a smaller data set and performed a full characterization on the larger i-LIDS PV3 data set. This shows a clear link between processing power consumption and event detection accuracy; compared with power-prioritized selection, automatic anomaly-driven mapping is 10% more accurate, but draws 12-W more power.

Acknowledgment

The authors would like to thank S. Robson for the work done at Thales Optronics and for his FPGA implementation

of the histogram generator for the CAR-HOG detector. The authors would also like to thank J. Thompson for helpful comments on the manuscript.

REFERENCES

[1] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool, "Pedestrian detection at 100 frames per second," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2012, pp. 2903–2910.

[2] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-based real-time pedestrian detection on high-resolution images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2013, pp. 629–635.

[3] D. Bacon, R. Rabbah, and S. Shukla, "FPGA programming for the masses," *ACM Queue*, vol. 11, no. 2, p. 40, Feb. 2013.

[4] C. C. Loy, T. Xiang, and S. Gong, "Detecting and discriminating behavioural anomalies," *Pattern Recognit.*, vol. 44, no. 1, pp. 117–132, Jan. 2011.

[5] B. T. Morris and M. M. Trivedi, "A survey of vision-based trajectory learning and analysis for surveillance," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 8, pp. 1114–1127, Aug. 2008.

[6] S. Atev, O. Masoud, and N. Papanikolopoulos, "Learning traffic patterns at intersections by spectral clustering of motion trajectories," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 4851–4856.

[7] C. Piciarelli and G. L. Foresti, "On-line trajectory clustering for anomalous events detection," *Pattern Recognit. Lett.*, vol. 27, no. 15, pp. 1835–1842, Nov. 2006.

[8] B. T. Morris and M. M. Trivedi, "Learning, modeling, and classification of vehicle track patterns from live video," *IEEE Trans. Intell. Transp. Syst.*, vol. 9, no. 3, pp. 425–437, Sep. 2008.

[9] A. Albiol, L. Sanchis, A. Albiol, and J. M. Mossi, "Detection of parked vehicles using spatiotemporal maps," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1277–1291, Dec. 2011.

[10] J. T. Lee, M. S. Ryoo, M. Riley, and J. K. Aggarwal, "Real-time illegal parking detection in outdoor environments using 1-D transformation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 7, pp. 1014–1024, Jul. 2009.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 886–893.

[12] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 4, pp. 743–761, Apr. 2012.

[13] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, pp. 1532–1545, Aug. 2014.

[14] V. A. Prisacariu and I. D. Reid, "fastHOG—A real-time GPU implementation of HOG," Dept. Eng. Sci., Oxford Univ., Oxford, U.K., Tech. Rep. 2310, 2009.

[15] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "FPGA-based pedestrian detection using array of covariance features," in *Proc. 5th ACM/IEEE Int. Conf. Distrib. Smart Cameras*, Aug. 2011, pp. 1–6.

[16] S. Bauer, S. Kohler, K. Doll, and U. Brunsmann, "FPGA-GPU architecture for kernel SVM pedestrian detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2010, pp. 61–68.

[17] C. Blair, N. M. Robertson, and D. Hume, "Characterizing a heterogeneous system for person detection in video using histograms of oriented gradients: Power versus speed versus accuracy," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 3, no. 2, pp. 236–247, Jun. 2013.

[18] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[19] B. Cope, P. Y. K. Cheung, W. Luk, and L. Howes, "Performance comparison of graphics processors to reconfigurable logic: A case study," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 433–448, Apr. 2010.

[20] Q. Wu, Y. Ha, A. Kumar, S. Luo, A. Li, and S. Mohamed, "A heterogeneous platform with GPU and FPGA for power efficient high performance computing," in *Proc. 14th Int. Symp. Integr. Circuits (ISIC)*, Dec. 2014, pp. 220–223.

[21] K. Deb, "Multi-objective genetic algorithms: Problem difficulties and construction of test problems," *Evol. Comput.*, vol. 7, no. 3, pp. 205–230, Fall 1999.

[22] Y. Yu and V. K. Prasanna, "Power-aware resource allocation for independent tasks in heterogeneous real-time systems," in *Proc. 9th Int. Conf. Parallel Distrib. Syst.*, 2002, pp. 341–348.

[23] H. Quinn, M. Leeser, and L. S. King, "Dynamo: A runtime partitioning system for FPGA-based HW/SW image processing systems," *J. Real-Time Image Process.*, vol. 2, no. 4, pp. 179–190, 2007.

[24] D. Llamocca and M. Pattichis, "A dynamically reconfigurable pixel processor system based on power/energy-performance-accuracy optimization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 3, pp. 488–502, Mar. 2013.

[25] C. G. Blair and N. M. Robertson, "Event-driven dynamic platform selection for power-aware real-time anomaly detection in video," in *Proc. Int. Conf. Comput. Vis. Theory Appl. (VISAPP)*, Lisbon, Portugal, Jan. 2014, pp. 54–63.

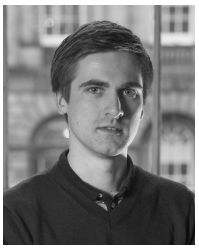
[26] N. Dalal, "Finding people in images and videos," Ph.D. dissertation, Dept. GRAVIR-IMAG, Inst. Nat. Polytechn. de Grenoble, Grenoble, France, 2006.

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Sep. 2009.

[28] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2. Jun. 1999, pp. 246–252.

[29] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proc. 17th Int. Conf. Pattern Recognit.*, vol. 2. 2004, pp. 28–31.

[30] Home Office Centre for Applied Science and Technology, *Imagery Library for Intelligent Detection Systems: The i-LIDS User Guide*, 4th ed. Horsham, U.K.: UK Home Office, 2011.
Horsham, UK



Calum G. Blair received the M.Eng. degree from Glasgow University, Glasgow, U.K., in 2009, and the D.Eng. degree from Heriot-Watt University, Edinburgh, U.K., in 2014.

He is currently a Post-Doctoral Research Fellow with the Institute for Digital Communications, School of Engineering, The University of Edinburgh, Edinburgh. His current research interests include real-time implementation of image and signal processing algorithms on parallel architectures, such as multicore, graphics processing units, and field-programmable gate arrays, and methods for obtaining high performance from embedded systems while minimizing power consumption.

Neil M. Robertson (SM'10) received the M.Sc. degree from Glasgow University, Glasgow, U.K., in 2000, and the D.Phil. degree from Oxford University, Oxford, U.K., in 2006.

He was involved in U.K. scientific civil service with DERA and QinetiQ from 2000 to 2007. He co-leads the EPSRC Edinburgh Centre for Robotics and the EPSRC/DSTL University Defence Research Centre. He is involved in human behavior recognition, computer vision, and multimodal sensor fusion with his research team. He is currently a Principal Investigator of the Vision Laboratory with Heriot-Watt University, Edinburgh, U.K., and an Honorary Fellow of the School of Engineering with The University of Edinburgh, Edinburgh.

Dr. Robertson held a prestigious 1851 Royal Commission Fellowship at the Robotics Research Group, Oxford University, from 2003 to 2006.