Energy Efficient Object Detection in Camera Sensor Networks

Tuan Dao*, Karim Khalil*, Amit K. Roy-Chowdhury*,
Srikanth V. Krishnamurthy*, Lance Kaplan†
*UC Riverside †U.S. Army Research Laboratory

***Abstract*—A wireless camera network can provide situation awareness information (e.g., humans in distress) in scenarios such as disaster recovery. If such camera sensors are battery operated, sending raw video feeds back to a central controller can be expensive in terms of energy consumption. Further, if all cameras were to use the optimal processing algorithm for object decision, they may also expend unnecessary energy. Stated otherwise, cameras that capture the same objects may not all have to use the optimal algorithm to achieve a desired accuracy, and this can save processing energy costs. In this paper, our objective is to design and implement a framework that can support coordination among cameras to deliver highly accurate detection of objects in an energy efficient way. The framework, which we call EECS (for energy efficient camera sensors), estimates the detection accuracy and energy costs incurred (both the processing and communication costs are taken into account) with each detection algorithm for each camera, and comes up with a choice of cameras for sending information pertaining to the object of interest. This set of cameras and the video processing algorithms that they must use, are chosen so as to minimize the energy expenditures, given a desired detection accuracy. We implement EECS on a camera network built with smartphones, and demonstrate that it reduces the energy consumption by up to 40% while ensuring a object detection accuracy of over 86%.**

I. Introduction

Timely and accurate detection of objects of interest (e.g., humans) is critical in many scenarios of interest. For example, in rescue and recovery missions following natural disasters, one might want to detect humans or animals in distress. Homeland security might be interested in automatically and proactively tracking unattended baggage in airports or bus stations. Today, camera sensors equipped with computational capabilities can be deployed in the field to provide situation awareness information in such scenarios. In fact, battery operated, low power embedded camera devices (e.g, the CMUcam series [1]) that can be used for such purposes are already emerging and on the market. Using common programming languages, these devices can be programmed to do multiple types of on-board processing tasks (e.g., object and face detection).

A network of such camera sensors can significantly improve the accuracy of object detection. With such a network, objects that might be obstructed or hidden from specific angles of view, can still be potentially detected. However, simply sending video feeds from all such camera sensors to a central controller that is responsible for operations (e.g., search and rescue) might not only be wasteful, but could result in unnecessary energy expenditures and hurt the longevity of the network. In addition, each camera could be trained to use highly optimal, domain specific, algorithms to process the captured video. However, the higher the fidelity of a processing algorithm, the higher the cost in terms of processing energy. Thus, when a plurality of cameras detect the same object, it might be unnecessary for all of the cameras to use energy expensive algorithms. Some of the cameras could use sub-optimal processing to save energy while ensuring that the detection accuracy does not take a big hit.

In this paper, our goal is to design a framework, called EECS, that can facilitate co-ordination among the cameras in such a network to realize significant energy savings, compared to cases where there is no such co-ordination, and yet achieve a high detection accuracy. The framework determines (a) which cameras are suitable for capturing objects of interest, (b) what domain specific algorithms to use for processing the captured video.

Challenges: In order to achieve our overarching goal, we need to tackle a set of key challenges. First, since the scenarios are likely to be unknown a priori, the most accurate video processing or detection algorithm for each camera sensor is not known. The problem is harder if we need to rank order the processing algorithms in terms of the accuracy they yield and the energy expenditures they incur for the scenario. This essentially requires the assessment of similarities between the video captured of an unknown scenario and a set of pre-installed training videos corresponding to a set of pre-determined scenarios; such an online comparison is very challenging for complicated and high dimensional signals like video feeds. Second, we require EECS to be able to identify a subset of camera sensors whose detection yields are together sufficient to achieve a desired accuracy. This ensures that EECS does not unnecessarily invoke all camera sensors and thus, helps reduce energy consumption. Third, we need to determine which camera nodes should utilize less accurate energy efficient detection algorithms, instead of using the most accurate (possibly more expensive) algorithm for processing the videos while still adhering to the accuracy requirements.

EECS in brief: EECS is designed to address the above challenges. To solve the first challenge, EECS leverages state-of-the-art video comparison algorithms to identify the most effective detection algorithm for each individual camera. In brief, each camera captures a short video feed and compares the feed with pre-loaded training videos to find the closest match; the algorithm that works best for the matching training video is then chosen. This process can be repeated when the environment changes. The process, which is referred to as "domain adaptation", has been used for efficient video comparisons [2]. Specifically, principle component analysis (PCA) is applied on the captured and training videos to remove the unimportant features, and to reduce the signal dimensionality; the PCA-processed signals are then projected onto a Grassmann manifold for comparison. The manifold is created in a manner that ensures that a small distance between two projected points in the manifold also indicates a high level of similarity between two associated video feeds.

The above approach however, only allows each individual camera to dynamically choose the most accurate algorithm to process the captured video feeds. EECS ranks the camera sensors based on individual accuracies and applies a novel greedy algorithm to choose a subset of cameras that jointly can achieve a predefined desired accuracy (thus addressing the second challenge). This requires EECS to be able to identify and aggregate detection information of the same objects from different views/cameras, and then assesses the detection accuracy based on the aggregated information. Subsequently towards addressing the third challenge, for each chosen camera, EECS determines whether each camera can use a less energy expensive algorithm that satisfies the desired detection accuracy requirement and if yes, chooses the less expensive algorithm for processing.

Novelty: To the best of our knowledge, EECS is the first to support coordination across a set of battery operated camera sensors towards reducing energy consumption while ensuring high accuracy of object detection. While the design of domain adaption for individual cameras has been studied in the computer vision community, coordination across cameras to determine the algorithms that different cameras should use to achieve a certain detection accuracy has not been considered before. Moreover, energy was not a consideration in determining the choice of this set of algorithms.

Evaluations: We implement EECS on a testbed of Android phones, which have pre-installed video feeds captured from overlapping cameras. We implement three different video processing algorithms on each of the camera views. The algorithms are adaptively chosen by EECS depending on the environment and requirements. Our evaluations show that EECS achieves both higher precision and recall than using the same algorithm to process all datasets. In addition, EECS's resource-aware algorithm selection approach helps to reduce up to 40% of the total energy consumption while still achieving $\approx 86\%$ of the highest accuracy (achieved when the most accurate algorithms are used at all individual cameras).

II. Related work

Object detection algorithms: While our work is applicable to object detection in general, we focus mainly on humans as the objects of interest. Different features (color, gradient, texture) and machine learning techniques (SVM, boosting) have been used in object detection [3], [4], [5], [6], [7]. However, each algorithm only works well in specific scenarios and conditions. In this work, we propose a framework for adaptively choosing an appropriate algorithm depending on the environment/condition.

Domain adaptation: Domain adaptation is used for learning classification rules for a target (e.g., indoor) dataset from a pre-trained source dataset [2], [8]. In EECS, domain adaptation is used to find the correspondence between features of the two datasets. This correspondence is then used to assess video similarity, and identify the most appropriate detection algorithm for an unknown incoming video feed.

Adaptive algorithm selection: Algorithm selection has been studied in several recent works for other problems. In [9], a model to predict the performance of different image segmentation algorithms is developed. In [10], pixels in an image are segmented into different regions, and different detection algorithms are applied for the different regions. In these works, the values of the selected features are used to determine which algorithms are used. In contrast, in our approach, the similarities between features from different video feeds are used to select the algorithm. The work in [11] is the closest to our work; the authors consider using different algorithms to detect humans in different video feeds. However, they only consider choosing the most efficient algorithm to process captured video feeds for a *single* camera. EECS, on the other hand, focuses on multi-camera settings in which camera diversity can be capitalized to reduce energy consumption.

Object detection using multiple cameras: Works such as [12], [13], [14] focus on detecting objects of interest using a network of camera sensors. However, they only employ a specific video processing algorithm to detect objects. On the other hand, EECS allows changing the detection algorithm adaptively as the environment changes to improve energy efficiency while ensuring detection accuracy.

Efficient video processing on mobile devices: Li *et al.* [15] propose data manipulation techniques to reduce memory access related energy consumption on mobile devices. Lee *et al.* [16] implement and study the energy consumption of different video encryption schemes on mobile devices. The authors in [17], [18], [19] study and improve the energy efficiency of video streaming applications for mobile devices. Improving the efficiency of object detection or other video processing algorithms is not our focus; we instead design a framework to choose the most energy efficient algorithm from a set of available algorithms while ensuring given accuracy requirements.

III. Video comparison using domain adaptation

When a video feed is captured, each camera needs to determine which video processing algorithm is most accurate. In order to do so, in EECS, the new feed is compared against a training set of videos that are pre-loaded onto the cameras. To determine matches between the new feed and the videos in the training set, an efficient video comparison technique is essential. This process is referred to as domain adaptation, i.e., determining which algorithm is best suited for the domain pertaining to the captured scene in the new feed.

Domain adaptation for a single camera has been studied in the computer vision community [2], [8] and can be used to determine the similarity between videos. In many cases, directly comparing video features in their original domains does not yield good results. For example, images taken in different conditions (indoor/outdoor, illumination, variations in size of an object from different views, etc.) can be quite different, but should actually be processed using the same algorithm to achieve the highest accuracy, as shown in [11]. In

Symbol	Meaning
α, β	Sizes of feature space and PCA subspace
t_i	Features of the training video item T_i
v_j	Features of the incoming video item V_j
k_1, k_2	Number of frames used to represent T_i and V_j
x_i, z_j	Basis of the PCA-projected subspaces of t_i and v_j
$\tilde{x_i}$	Orthogonal complement to x_i, namely, $\tilde{x_i}^T x_i = 0$
$\theta(y)$	Geodesic flow function
U, V, Σ_1, Σ_2	Matrices used to compute $\theta(y)$ and W_{ij}
W_{ij}	Geodesic kernel, used to compute distance between x_i and z_j

TABLE I: List of symbols used in computing video similarity

such cases, the similarity between two video feeds[1] is much more noticeable if the features in the feeds are projected on to a common subspace. We use image key-points, and the histogram of oriented gradients (HOG) as features of the image frames in a video feed for comparison; the chosen features, to be used in EECS, will be described in detail in section V.

The key idea in domain adaptation, is to project the two video feeds (a training video and an unknown video feed) onto a common subspace, in which similar patterns between the videos can be better identified. Here, we choose to project the training and the incoming videos (also referred to as data items) onto a Grassmann manifold, as in [2]. The geodesic flow curve is the *shortest path* that links two projected items on the manifold, and represents a measure of similarity of the data distributions on the manifold. If two items have similar distributions on the Grassmann manifold, the same video processing or detection algorithm should be applied on the two video feeds [11].

In more detail, we formulate the problem of assessing the similarity between two videos as follows [2]. Let the features to be compared in the training and captured videos, T_i and V_i be $t_i \in \mathbb{R}^{k_1 \times \alpha}, v_j \in \mathbb{R}^{k_2 \times \alpha}$, respectively. Here, k_1, k_2 are the number of key frames (images) chosen in T_i and V_j, respectively, to represent the entire video feeds to reduce the computational overhead. In addition, α is the dimension of the feature vector of each chosen key frame. In other words, each video feed is represented as a set of images, where each image is then represented as a feature vector in the $\mathbb{R}^\alpha$ space.

Using principal component analysis (PCA), we project all the images in T_i onto a $\mathbb{R}^\beta$ subspace in which the variances of data are maximized; typically $\beta < \alpha$. The basis of such a subspace consists of β orthogonal α-dimensional basis vectors. Let $x_i \in \mathbb{R}^{\alpha \times \beta}$ be the basis of the original subspace and $z_j \in \mathbb{R}^{\alpha \times \beta}$ be the basis of the subspace obtained when applying PCA on v_j. Let $Gr(\beta, \mathbb{R}^\alpha)$ be a special space that contains all the subspaces of size β in $\mathbb{R}^\alpha$; called a Grassmann manifold of $\mathbb{R}^\alpha$. Then, both the subspaces represented by x_i and v_j lie on $Gr(\beta, \mathbb{R}^\alpha)$. Let $\tilde{x_i} \in \mathbb{R}^{\alpha \times (\alpha - \beta)}$ be the orthogonal complement to x_i, namely $\tilde{x_i}^T x_i = 0$, where x^T denotes the transpose of matrix x.

Given the Grassman manifold, the geodesic flow connecting x_i and z_j on the manifold, is defined as [2]:

$$\int_0^1 (\theta(y) t_i)^T (\theta(y) v_j) dy = t_i^T W_{ij} v_j. \tag{1}$$

The left hand side of (1) provides the definition of the geodesic flow, whose value can be computed using the right side of the equation. $\theta(y)$ is the geodesic flow function, parameterized by a continuous variable $y \in [0, 1]$ [2].

The geodesic flow can be computed by computing the kernel function W_{ij} between the two feature vectors t_i and v_j. W_{ij} is defined as:

$$W_{ij} = [x_i U \quad \tilde{x_i} V] \begin{bmatrix} \Lambda_1 & \Lambda_2 \\ \Lambda_2 & \Lambda_3 \end{bmatrix} \begin{bmatrix} U^T x_i^T \\ V^T \tilde{x_i}^T \end{bmatrix}, \tag{2}$$

where U, V are the left and right singular matrices when applying singular value decomposition (SVD) to $x_i^T z_j$ and $\tilde{x}_i^T z_j$, respectively. Further, let Σ_1 and Σ_2 be the diagonal matrices of such SVDs; the values of Λ_1, Λ_2, and Λ_3 matrices are computed from both Σ_1 and Σ_2 [2]. The kernel function W_{ij} provides an effective way to compute the inner product of high dimensional vectors t_i and v_j, which is widely used to compute the similarity between vectors [20].

The kernel distance between the two video feeds T_i and V_j (i.e., between features of two sets of images t_i and v_j) on the manifold is computed based on the geodesic flow connecting them as [21]:

$$K(T_i, V_j) = t_i^T W_{ij} t_i + v_j^T W_{ij} v_j - 2 t_i^T W_{ij} v_j, \tag{3}$$

where $K(T_i, V_j)$ is a $k_1 \times k_2$ matrix representing the individual distances on the manifold from each image in T_i to each image in V_j.

We define the total distance between the two video feeds on the manifold to be the mean of all the kernel distances between the individual images from the two feeds:

$$M_d(T_i, V_j) = \frac{1}{k_1 k_2} \sum_{m_1} \sum_{m_2} K_{(m_1, m_2)}(T_i, V_j), \tag{4}$$

where m_1, m_2 are integers in $\{1, \cdots, k_1\}, \{1, \cdots, k_2\}$, respectively, and $K_{(m_1,m_2)}(T_i, V_j)$ is element (m_1, m_2) of the matrix $K(T_i, V_j)$.

Finally, we define the similarity of the two videos T_i, V_i as:

$$Sim(T_i, V_j) = e^{-M_d(T_i, V_j)}. \tag{5}$$

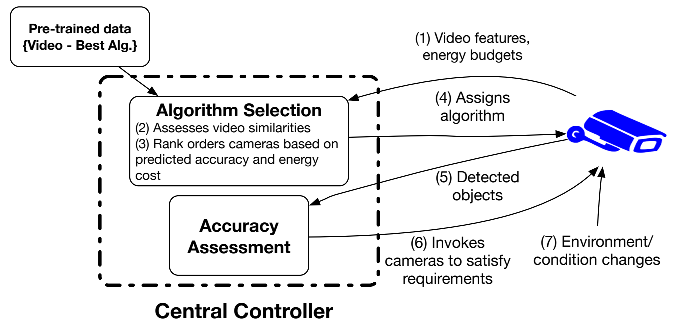
Notice that $Sim(T_i, V_j) \in [0, 1]$, for $M_d(T_i, V_j) \geq 0$. A higher distance corresponds to a lower similarity value. Further, the similarity approaches 0 exponentially fast beyond some certain threshold (e.g., when $M_d(T_i, V_j) \geq 4$). In such cases, the video feeds are considered dissimilar.

IV. EECS SYSTEM DESIGN

In this section, we describe the design of our camera coordination framework EECS. EECS consists of two main components: camera sensors and a central controller (we also use the terms cameras and controller). In EECS, the central controller collects visual features (discussed later in section V) from the camera sensors and use these to determine the most effective detection (video processing) algorithm, for each camera, in terms of accuracy and energy consumption. In addition, EECS decides what combination of views yields the desired object detection accuracy, while ensuring that the energy drain at the camera adheres to a set energy budget. In EECS, video analytics and algorithm selection happen at the

[1]We use the terms video feed and video item interchangeably.

[2]The list of symbols used in this section is summarized in Table I.



Pre-trained data {Video - Best Alg.}
(1) Video features, energy budgets
Algorithm Selection
(2) Assesses video similarities
(3) Rank orders cameras based on predicted accuracy and energy cost
(4) Assigns algorithm
(5) Detected objects
Accuracy Assessment
(6) Invokes cameras to satisfy requirements
(7) Environment/ condition changes
Central Controller

Fig. 1: EECS system for adaptively choosing detection algorithms in a camera network.

controller to avoid storing information about training video feeds and executing processing-expensive, domain adaptation at each battery-operated camera sensor. Here, as is typically the case, we assume that the controller does not have energy constraints and can easily perform these required operations.

Each camera individually executes a detection algorithm to detect the presence of objects (e.g. humans) in the scene. The detection accuracy of a certain algorithm depends on how well it matches the environmental conditions, which are dictated by attributes such as brightness and indoor versus outdoor, etc. At the same time, different algorithms consume different amounts of energy. The controller collects information relating to the detected objects as well as residual energy information from the camera sensors. Based on the assessment of the achieved global detection accuracy and the energy budgets and expenditures at each camera, the controller adaptively invokes different sets of cameras and/or different detection algorithms to meet both the accuracy and energy requirements. Fig. 1 depicts the functional view of system with its different components. In the following, we describe the details of EECS.

Let $\mathcal{T} = \{T_1, T_2, \cdots, T_N\}$ be the set of training videos at the controller. Let $\mathcal{A} = \{A_1, A_2, \cdots, A_H\}$ be the set of available detection algorithms pre-installed at each individual camera. Each camera sensor $S_j \in \mathcal{S}$ of M cameras has an energy budget B_j, which is a function of the required operation time as well as other processing parameters (such as number of frames processed per second). In addition, each camera has a communication cost C_j, which depends on the link quality from the camera to the central controller and is independent of the detection algorithm assigned to the camera[3]. Let $\mathcal{V}$ be the set of captured video feeds, where $V_j \in \mathcal{V}$ is the video feed captured at camera S_j. The goal of EECS is to identify a subset of the camera sensors $\mathcal{S}' \subseteq \mathcal{S}$ and the corresponding video processing algorithm $A'_j \in \mathcal{A}$ to be used at *each* camera $S_j \in \mathcal{S}'$ such that energy consumption is minimized subject to minimum global detection accuracy $\mathbf{D}$, to be defined in Section IV-C, and maximum energy constraints $c(A'_j) + C_j \leq B_j$. Here, $c(A'_j)$ represents the computation cost for algorithm A'_j. We develop a greedy algorithm to address this combinatorial optimization problem.

[3]Specifically, C_j depends on the resolution of the captured video, and the available bandwidth between the camera sensor and the central controller. This can be estimated using tools such as iPerf [22], by transferring some sampled frames and recording the consumed energy.

A. *Offline training*

A key task of the controller is to rank order the video processing algorithms based on their detection accuracies and identify the most accurate algorithm for the captured video feed V_j with regards to each individual camera sensor S_j. EECS performs a video comparison of the incoming video with the training videos using the domain adaptation technique, described in section III.

First, the controller applies each available detection algorithm to process each training item, and measures the computational cost and the detection accuracy achieved (a total of $H \times N$ combinations). Processing energy costs of the algorithms are estimated by applying each algorithm to a few sampled frames and recording the consumed energy, using tools such as PowerTutor [23]. For the detection accuracy, the controller measures the precision and recall values for each algorithm. The precision value is the number of correctly identified objects from among the detected objects, while the recall is the number of detected objects from among the objects actually in the scene. We consider scenarios where both precision and recall are important to the detection performance. Since these two metrics could be conflicting, we consider the f_score value [24], which balances both metrics and is usually used to assess the accuracy of a detection algorithm. It is computed as $f\_score = 2 \times \frac{recall \times precision}{recall + precision}$. For each training item, a ranked list of algorithms is then constructed based on the f_score. For each training video $T_i \in \mathcal{T}$, the most accurate detection algorithm satisfying the energy constraint, labeled $A_i^* \in \mathcal{A}$, is identified.

B. *Resource-aware algorithm selection*

In this subsection we describe how the information gathered at the central controller from the camera sensors is used to adaptively choose the detection algorithm for each camera sensor.

1) Uploading video features: When the camera sensors start up, or when surrounding environmental changes are detected, each sensor S_j extracts and uploads features, such as image key points and HOG, of the captured video feed V_j to the controller (details discussed in section V). Further, each camera notifies the controller about its energy budget B_j. Note that, detection of environmental changes is not in the scope of this paper.

2) Rank ordering the detection algorithms: Once the central controller receives video features V_j from camera sensor S_j, it determines the video similarities between the input and the items in its training set, and identifies the closest training item $T_i^* \in \mathcal{T}$ that is most similar to V_j using Equation (5). Because EECS uses state-of-the-art video comparison techniques, a high similarity between T_i^* and V_j indicates that the two videos should be processed by the same algorithm [11]. Thus, algorithm A_i^*, associated with T_i^*, will be the most accurate algorithm that can be used to process V_j. Further, the process ensures that the rank ordering of the detection algorithms will also be similar between the videos.

3) Choosing a subset of cameras: Periodically, for a short *accuracy assessment* duration (e.g., 100 frames), each camera employs all the available detection algorithms satisfying their energy budget and then transmit the detection metadata to the controller. This information is then used to estimate the best possible global accuracy that can be achieved [4]. The cameras are then rank ordered based on their individual accuracies in the list $\mathcal{S}_o = [S_1, S_2, ..., S_m]$.

Next, the central controller iterates over subsets of cameras in $\mathcal{S}_o$, estimating the global accuracy when cameras are activated sequentially from the list $\mathcal{S}_o$ in order, until the desired global accuracy is satisfied. The set of chosen cameras is denoted by $\mathcal{S}' \subset \mathcal{S}$. This approach ensures that EECS does not invoke all the camera sensors unnecessarily, but only invokes a sufficient set of cameras to satisfy the detection accuracy requirements while conserving energy.

4) Choosing detection algorithms: In the previous step, a set of cameras that satisfies the required accuracy using the associated most accurate (i.e., best) detection algorithms $A_j^*, \forall j \in \{1, 2, \cdots, M\}$ are chosen. Here, we seek to further reduce the energy usage at the camera sensors while still satisfying the detection accuracy requirements.

First, the controller browses the list $\mathcal{S}'$ in reverse order to consider the cameras with the lower accuracies first, towards reducing energy expenses. Then, for each camera, the controller checks whether a different lower energy algorithm can be used, while still retaining the required global accuracy $\mathbf{D}$. This reduction in accuracy is a direct consequence of the algorithm chosen at that camera. In other words, at each step, the accuracy with regards to the other cameras is not affected. Note that, in order to reduce the number of alternatives that need to be explored, EECS only pays attention to algorithms that have higher $\frac{f\_score}{\text{energy cost}}$ values compared to the most accurate algorithm. It selects this algorithm for the corresponding camera and feeds back this information to the camera sensor. If such an algorithm is not found, then this process stops. Otherwise, the process continues with the next camera in $\mathcal{S}'$.

Next, the set of cameras $\mathcal{S}'$ is used, along with the selected detection algorithms, until the next *re-calibration interval* (e.g., after 500 frames). At that point, a new accuracy assessment process starts, and a new set of camera sensors and detection algorithms might be chosen.

C. Global detection accuracy estimation

In our algorithm, the central controller assesses the global detection accuracy, given the accuracy assessments from the different camera sensors. This process is explained in detail as follows.

In order to assess the global accuracy, the controller needs first to identify the same object (e.g., human) captured from different cameras/views, and estimate the achieved accuracy pertaining to that object. Correctly aggregating the same detected areas (representing objects) from multiple views allow EECS to correctly identify the total number of objects that has been detected in the scene, since those areas will be counted as a single object. Otherwise, the same detected human might be counted multiple times, which leads to an incorrect assessment of the global detection accuracy. However, this re-identification of the same object from multiple camera views is challenging in itself; this is because images of the same object can be quite different if viewed from different angles. In the following, we discuss how detection metadata are used to aggregate multiple views of the same object.

Aggregating detection metadata from multiple cameras: For each detected object in an image, object detection algorithms provide the location (rectangular bounding box) of the area on the image as well as a score reflecting how confident the algorithm is, with regards to the area representing an object of interest (for example, see the detection algorithm in [3]). Thus, for each detected area, the sensors extract and upload metadata of that area representing a potential object. Specifically, this metadata includes: (i) the location of the area in the image, (ii) color features of the area, and finally (iii) a confidence measure that the detected area is an actual object of interest. This metadata is then used for object re-identification as follows.

First, a set of landmark points on the ground are chosen in the real world coordinate system. The locations of these landmarks are then identified in the captured images of each individual cameras. Using the correlation between the locations of the landmarks in the images captured by two cameras facilitates the building of a mapping function (called a homography) between the "ground planes" of the two cameras (e.g., by using RANSAC [25], which produce very accurate results). Such homographies are built offline and need to be recalibrated only if the camera geometry (e.g., orientation, zoom) changes. Once such a homography is constructed, for each detected area in an image, the central controller extracts the center of the bottom edge in the frame, which is supposed to be on the ground, and then projects that center point onto the ground plane of other camera views to identify detected areas of the same object in the other camera views.

Next, the controller uses the color features of the detected areas to reduce the false matches due to imperfect homography matching. Specifically, in EECS, we extract the Mean Color feature [26] of a detected area, and then use PCA to reduce the number of features. Then, the Mahalanobis distance [27] is used to compute the distance between the color features of two detected areas by different cameras that were pre-matched by homography mapping. If the distance is within a certain threshold, we consider the two detected regions to correspond to the same object.

Finally, for objects that are re-identified using location of the area on the image and color features, we need to combine their confidence scores to have a single confidence measure for the corresponding object. We discuss how the central controller fuses object detection scores from multiple cameras

Assessing global detection accuracy: First, we denote the area on the image for each detected object i, using the algorithm running on camera S_j, by R_{ij}. Each area R_{ij} has a corresponding detection probability P_{ij}[5], representing the detection precision associated with that area. Thus, P_{ij} indicates the probability that the area R_{ij} is actually an object of interest, while $(1 - P_{ij})$ is thus the false positive probability of object i on camera j. The combined *true positive* detection probability P_i is then computed as the complementary probability that *all* cameras S_j yield false positive detections Thus, P_i, is then

[4]The metadata consists of features extracted from the video (e.g., color features). Metadata and global detection accuracy will be defined formally in Section IV-C.

[5]Object detection scores can be converted into detection probabilities via an offline training process.

given by

$$P_i = 1 - \prod_j (1 - P_{ij}). \tag{6}$$

In reality, ground truth information may not be available at operation time, and involving a human in the loop to manually verify the detection is assumed to be not possible or expensive (for instance, if the human has to continuously provide feedback on the results). Thus, it is not possible to quantify achieved accuracy measures such as precision and recall to be used in updating camera and algorithm selection.

Thus, in this paper, global detection accuracy is characterized by two measurable quantities: (i) the number of objects on the field jointly detected by the cameras (after re-identification), and (ii) the combined detection probability of each detected object. To bound detection accuracy of system, the controller periodically triggers each camera S_j to use the highest accuracy algorithm A_j^* to compute a "baseline" global detection accuracy, quantified by the number of detected objects, and the average detection probability. If the gap between the detection accuracy achieved by current camera and algorithm selection and that achieved by the baseline is large, more cameras and/or more expensive algorithms are invoked.

Specifically, we let N^* and P^* be the number of detected objects and the average detection probability (of all detected objects), respectively, when the *best* algorithms are used at *all* the cameras. Here, the probability for each object is computed by Equation 6. The desired accuracy is then defined proportionally to the values of N^* and P^*. Specifically, let $\mathbf{D} = [D_n, D_p]$, where D_n and D_p be the desired number of detected objects and the desired mean value for the detection probability, respectively. We then require that $D_n \geq (\gamma_n \times N^*)$, and $D_p \geq (\gamma_p \times P^*)$; the values of γ_n and γ_p can be changed to influence the desired accuracy.

Summary: Fig. 2 summarizes the operations and interactions between video sensors and the central controller. The individual camera sensors capture video feeds, extract specific features and send these features along with their residual energy information, to the central controller. The central controller does the video analytics to select a sub-set of cameras, and the video processing algorithms that must be used at these cameras, to achieve a good trade-off between detection accuracy and energy expenses.

V. Implementation

In this section, we describe the detailed implementation of EECS, which consists of two main components viz., camera sensors and a central controller that gathers the outputs from these cameras.

Note that, in implementing EECS, we focus on "humans" as objects; this has significant importance in rescue, tactical and homeland security missions. The techniques can however, be used to detect other types of objects; only the detection algorithms used at the camera sensors need to be replaced.

A. The camera sensors

We implement the camera sensors using Asus Zen II Android smartphones. Each node is pre-installed with 4 different

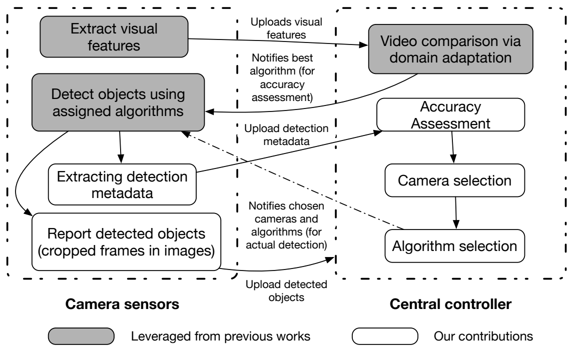

Extract visual features
Uploads visual features
Video comparison via domain adaptation
Notifies best algorithm (for accuracy assessment)
Detect objects using assigned algorithms
Accuracy Assessment
Upload detection metadata
Extracting detection metadata
Camera selection
Notifies chosen cameras and algorithms (for actual detection)
Report detected objects (cropped frames in images)
Algorithm selection
Upload detected objects
Camera sensors
Central controller
Leveraged from previous works
Our contributions

Fig. 2: Interactions between the sensors and central controller

human detection algorithms: HOG[6] [3], ACF [4], C4 [6] and LSVM [5]. We use OpenCV to implement HOG, LSVM, and ACF, based on the source code provided by the authors. For C4, we use the source code (in C++) provided by its authors.

First, each captured video feed is represented by 100 image frames. For each frame, we extract the HOG features [3] and SURF (speeded-up robust features) key-points[7] [28] of the image using OpenCV. The HOG features are represented by a 3780-dimension feature vector. For the SURF key-points, we use the bag of words (BoW) approach [29]. Specifically, each SURF key-point is represented by a 64-dimensional vector, called the key-point descriptor. Once a training set is chosen (more details about data sets are provided later in section VI), key-point descriptors of the training images are extracted, and then partitioned into predefined k clusters using the k-means clustering algorithm. Each such cluster centroid is called a *visual word* in the vocabulary.

In EECS, a vocabulary of 400 words is built from images of 12 training video feeds. Subsequently, for any given image, the key points of the image are extracted, and each key point is then mapped to the nearest cluster centroid (visual word) [29]. The BoW representation of an image, regardless of the image size and number of key-points, is thus a 400-bin histogram, in which the value of each bin is the number of key-points mapped on to the associated visual word. Thus, each image frame in a incoming video feed is actually presented by a fixed 4180-dimension feature vector (combining the HOG feature and the BoW representation), or about 16KB. The camera nodes then upload the features of the set of chosen image frames of the captured video feed to the central controller for video comparisons and subsequent, processing (detection) algorithm selection.

Once a detection algorithm is assigned by the central controller, the camera sensors detect the presence of objects of interest, and upload the information relating to the detected areas (the locations and the color features of the objects in the frame) to the controller for accuracy assessment and re-calibration. We also utilize OpenCV to extract features from the detected areas on the smartphones camera sensors. The amount of detection metadata transmitted to the controller depends on the number of objects detected. For each detected

[6]We use the term HOG for both the feature (histogram of gradient) and the algorithm that leverages the feature.

[7]Key-points are small patches of an image that differ significantly from the surrounding areas (in the image).

object, the metadata includes 8 bytes representing the position (rectangular bounding box) of the object in the image, 4 bytes for the detection probability, and finally, 160 bytes representing the 40-dimensional color feature. Thus, a total of 172 bytes are transmitted for each detected object.

B. The central controller

The central controller is implemented on a Linux server. It contains a pre-installed training set consisting of different video items (details in section VI). The accuracy ($precision$, $recall$ and f_score values) of each algorithm on each training item is computed.

To compute the similarity between the training videos and the camera sensor frames sent to the controller, the source code provided by the authors of [2] is used to compute video distances on the Grassmann manifold. The controller then follows the framework described in Section IV-B to select a subset of cameras and the associated detection algorithms to achieve the desired accuracy.

VI. Evaluations

In this section, we evaluate both the accuracy and energy efficiency of EECS.

Training and test datasets: We use the following publicly available datasets in our evaluation; each of the datasets consists of video feeds captured from 4 overlapping cameras:

- Dataset #1 is the "lab sequences" dataset, provided by EPFL [30]. This dataset consists of indoor video feeds in an empty room setting. In each video there are 6 people walking in the room. The resolution of the video set is 360x288.
- Dataset #2 is the "chap" dataset, provided by Graz University [31]. This dataset also consists of indoor video feeds in a lab setting, in which there are 4-6 people walking in the room. There are furniture items in the lab which might cause false positives in terms of detection; thus this dataset has lower precision than the other datasets. The resolution of this dataset is 1024x768. Thus the energy cost to process this dataset is expected to be higher than in the other datasets.
- Dataset #3 is the "terrace sequences" dataset, also provided by EPFL [30]. This dataset contains outdoor video feeds, with 8 people walking on an empty terrace of a building. The resolution is 360x288.

Since each set contains video feeds captured simultaneously from 4 overlapping cameras, there are 12 videos feeds in total. Each of those video feeds is approximately 3000 frames long. We use the first 1000 frames in each video feed as the training video. The remaining 2000 frames from each video feed are used as test data.

Ground truth information: All the datasets we use contain ground truth information about human locations in the scene. In particular, the 3D locations (in the real-world coordinates) of where the humans stood in the scene are marked. Further, for each video feed in the dataset, the homography used to transform between coordinates on the ground plane in the image and real-world coordinates is provided[8]. Using such homography, those 3D locations are converted to 2D locations in each video frame. This is then compared with the results of the detection algorithm to estimate accuracy in our evaluation. For datasets #1 and #3, ground truth information is available every 25 frames, whereas the ground truth is available every 10 frames for dataset #2.

[8]Such a homography is built by marking the landmarks in the field and in the images, as described in IV-B.

Re-identifying detected objects across cameras: Using the provided homography information described above, EECS can re-identify and aggregate the same objects detected by different cameras. Color features of the matched objects are then verified to reduce false positives, as described in [26]. By using these two techniques, EECS is able to re-identify objects with a high precision (more than 90%) in all the datasets.

Computing energy costs and budget: For each sensor, we apply each algorithm to each of the videos to learn the processing cost of the algorithm. The process is repeated over 1000 frames to get the average value. We transfer 1000 frames, compressed using JPEG format, using WiFi in good conditions to estimate the communication cost. In EECS, sensors only transfer cropped image frames containing the detected objects to the controller. The amount of transferred data thus varies across frames since the number of detected objects and the sizes of the different objects also vary. Thus, to estimate the communication cost, we assume the whole frame is transferred, and monitor the consumed energy. Doing so ensures the actual communication cost will never be higher than our estimated value.

Finally, the energy budget is computed by first defining an expected operation time (e.g., 6 hours) and an expected frame rate (e.g., image frames are processed every 2 seconds). Given these, the number of frames needed to be processed during the operation time (e.g., how often batteries need to be recharged or replaced) can be computed. Subsequently, the residual energy capacity is divided by the number of frames to compute the energy budget for each frame. In the evaluations that are presented later, we actually use different budget values to evaluate how EECS adaptively chooses different algorithms under different given budget constraints.

A. Estimating the detection accuracy

Each camera sensor is pre-installed with 4 different detection algorithms. Video feeds available are split into training segments and test segments. We apply all the algorithms to each of the training video segments to characterize the accuracy of each algorithm on each training segment. Each object detected is assigned a *detection score* by the algorithm, reflecting a measure of confidence in detection. We discard areas on a frame with very low detection score below a cut-off detection score threshold d_t. Different cut-off thresholds correspond to different values of f_score. For example, a higher threshold will disregard detection areas with lower scores to reduce the false positive rate (increase precision), but on the other hand, might also cause correctly detected areas with lower scores to be ignored and thus reduce recall. Thus, even though we can increase precision or recall individually, we choose a threshold d_t which maximizes the f_score value. In other words, for each combination of an algorithm and a training video segment, we record the highest possible accuracy (as measured by f_score) the algorithm can achieve on that segment.

Alg.	Threshold	Recall	Precision	F-score	Energy cost/frame(J)	Processing time/frame (s)
HOG	0.5	0.48	1.0	**0.66**	1.08	1.5
ACF	2	0.34	0.95	0.505	0.07	0.1
C4	0	0.46	1	0.63	4.92	2.4
LSVM	-1.2	0.89	0.9	0.89	3.31	6.2

TABLE II: Accuracy of different algorithms on dataset #1, camera #1, frame 0→1000, used as a *training* video item.

Alg.	Threshold	Recall	Precision	F-score	Energy cost/frame(J)	Processing time/frame (s)
HOG	0.6	0.8	0.42	0.55	9.86	3.4
ACF	20	0.83	0.89	**0.86**	0.315	0.4
C4	0.5	0.70	0.70	0.70	5.56	6.8
LSVM	-0.2	0.84	0.83	0.84	25.06	32.2

TABLE III: Accuracy of different algorithms on data set #2, camera #1, frame 0→1000, used as a *training* video item.

Alg.	Threshold	Recall	Precision	F-score	Energy cost/frame(J)	Processing time/frame (s)
HOG	0.5	0.6	0.99	**0.74**	1.07	1.8
ACF	2	0.52	0.91	0.66	0.07	0.1
C4	0	0.534	0.974	0.69	4.82	2.3
LSVM	-1.2	0.975	0.892	0.93	3.2	6.4

TABLE IV: Accuracy of different algorithms on dataset #1, camera #1, frame 1001 → 2950, used as a *test* item.

The video feeds are split into two segments. The first segment, 1000 frames, is used as training items. Tables II and III show the efficiencies of the detection algorithms on the training items extracted from video feeds captured from camera #1, in dataset #1 and in dataset #2, respectively. We also apply the algorithms on the test video segments using the same threshold values learned from the training segments. Table IV shows the accuracy of the detection algorithms on the feed captured from camera #1 in dataset #1, from frame 1001 to 2950. The energy costs shown in these tables include the processing costs of the corresponding algorithm as well as the algorithm-independent communication cost to transfer the images of detected objects to the central controller, as described earlier in this section.

In Tables II and IV, we use two different segments from the same video which is captured by camera #1 on dataset #1. The tables show that the LSVM algorithm, even though has a very high f_score value, also has very high energy cost and processing time; thus, it can be expensive to use on a battery driven mobile platform. Thus, we will not consider LSVM in the remainder of this section. Consequently, for video feeds on dataset #1, camera #1, HOG will be considered the most accurate detection algorithm.

B. Evaluating video similarity using domain adaptation

Extracting features from an entire video feed is expensive. Here, we only extract the features (HOG, and BoW) of a 100 consecutive frames. To reduce the bias, the frames are randomly selected from each video feed and the process is repeated 5 times. We then report the average value of the similarity.

Table V shows the similarities between the training and test video items, computed as described in (5). In the table, $T_{x.y}$ (or $V_{x.y}$) indicates the training (or test) video item from dataset $\#x$ and captured by camera $\#y$. It is shown that, using the manifold distance, in all the cases, we are able to match a test item to the training item corresponding to the same dataset and captured by the same camera. This observation is also confirmed by the results in tables II and IV. These tables show that video items belonging to same camera and same dataset have the same most accurate detection algorithm as well as the same "order" of the algorithms in terms of accuracy.

C. Benefit of adaptively choosing the detection algorithms

Next, we show the benefits of adaptively choosing different algorithms to process different video feeds. Fig. 3 shows the highest detection accuracy when different algorithms are used to process the video feeds captured by camera #1 in both dataset #1 and dataset #2.

Assuming that the environment changes (from dataset #1 to #2), if the same algorithm is still used, the highest f_score the system can achieve by using one detection algorithm is 0.70 (using HOG algorithm) for both datasets. However, if the system adaptively uses the best algorithm for each dataset (specifically, HOG for dataset #1 and ACF for dataset #2), the highest f_score achieved is 0.81.

More importantly, adaptively choosing the most accurate algorithm helps increase the recall and precision values *simultaneously*. Specifically, if HOG algorithm is used, the achieved recall is 0.71, which is close to 0.73 of the adaptive approach. However, the precision is much lower, 0.68, compared to 0.91 achieved when the adaptive approach is used (corresponding to a higher false positive rate compared to the adaptive approach). Similarly, when ACF is used, the precision is good, however, the recall is significantly lower than the adaptive approach. In other words, the false negative rate is high. Thus, using an adaptive approach helps reduce both the false negative and false positive rates simultaneously.

D. Should the highest accuracy algorithm always be used?

Based on video comparison, EECS can identify the most accurate detection algorithm for an incoming video feed. However, should the most accurate algorithms *always* be used to process the video feeds?

Fig. 4 shows the trade-off between the achieved accuracy (in terms of the number of correctly detected humans) and energy costs when processing the 4 video feeds in dataset #1. We show the values when 2 cameras are used: (i) 2HOG: both cameras used HOG (the most accurate, yet expensive, algorithm), (ii) 2ACF: both cameras used ACF (less accurate, yet energy efficient, algorithm), (iii) HOG+ACF: one camera used HOG and the other used ACF; and when 4 cameras are used: (iv) 4HOG: all 4 cameras used HOG, (v) 4ACF: all 4 cameras used ACF, and finally, (vi) 2HOG+2ACF: two camera used HOG while the other two cameras used ACF.

The x-axis shows the recall achieved (the number of humans detected among the humans appearing in the scene)[9], while the y-axis shows the energy consumption for each case.

It is shown that, depending on the desired accuracy, a less accurate algorithm can be used to achieve a significantly lower energy consumption, but with a relatively small accuracy

[9]For dataset #1, the precision is ≥ 0.95 for all the algorithms (Table II). Thus we only pay attention to the recall values in this case.

Test set/ Train set	$V_{1.1}$	$V_{1.2}$	$V_{1.3}$	$V_{1.4}$	$V_{2.1}$	$V_{2.2}$	$V_{2.3}$	$V_{2.4}$	$V_{3.1}$	$V_{3.2}$	$V_{3.3}$	$V_{3.4}$
$T_{1.1}$	**0.78**	0.56	0.53	0.56	0.47	0.49	0.48	0.45	0.46	0.48	0.49	0.44
$T_{1.2}$	0.55	**0.77**	0.54	0.60	0.46	0.48	0.48	0.49	0.48	0.49	0.46	0.40
$T_{1.3}$	0.54	0.54	**0.76**	0.53	0.45	0.47	0.50	0.45	0.48	0.49	0.49	0.41
$T_{1.4}$	0.56	0.61	0.54	**0.76**	0.47	0.50	0.51	0.48	0.48	0.53	0.48	0.40
$T_{2.1}$	0.39	0.39	0.38	0.38	**0.79**	0.45	0.48	0.43	0.37	0.37	0.39	0.34
$T_{2.2}$	0.44	0.47	0.45	0.47	0.48	**0.75**	0.51	0.45	0.43	0.43	0.46	0.39
$T_{2.3}$	0.41	0.45	0.43	0.46	0.49	0.51	**0.81**	0.47	0.45	0.41	0.41	0.37
$T_{2.4}$	0.38	0.43	0.39	0.41	0.45	0.44	0.47	**0.76**	0.41	0.36	0.39	0.37
$T_{3.1}$	0.50	0.54	0.51	0.53	0.43	0.45	0.48	0.49	**0.69**	0.48	0.46	0.45
$T_{3.2}$	0.44	0.47	0.44	0.49	0.41	0.41	0.45	0.39	0.40	**0.69**	0.43	0.38
$T_{3.3}$	0.48	0.48	0.49	0.49	0.47	0.47	0.44	0.43	0.41	0.46	**0.74**	0.49
$T_{3.4}$	0.45	0.43	0.45	0.43	0.42	0.41	0.43	0.42	0.42	0.40	0.51	**0.75**
$T_{x.y}$, $V_{x.y}$ denote training and test video feed captured by camera #y in dataset #x, respectively												

TABLE V: Video similarities computed using the manifold distance

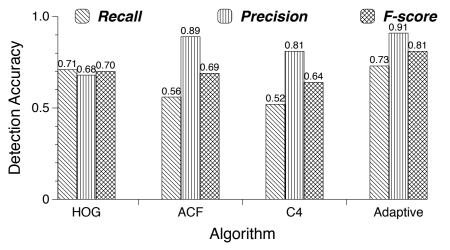


Fig. 3: Achieved accuracy with different detection algorithms, dataset #1 and #2

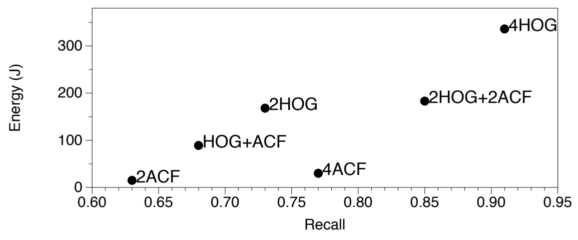
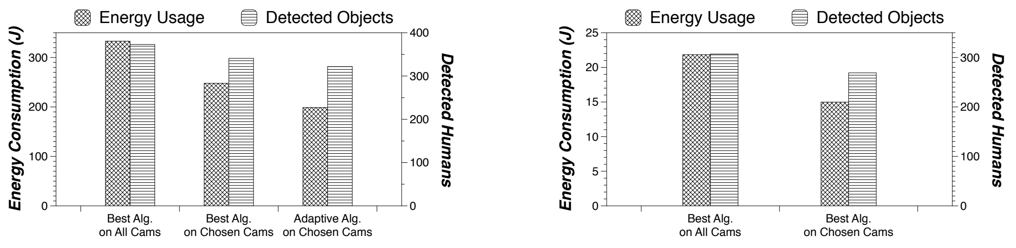


Fig. 4: Trade-off between accuracy and energy cost to process dataset #1



(a) Energy budget ≥ 1.08

(b) Energy budget $\in [0.07, 1.08)$

Fig. 5: Detected humans vs. energy consumption for dataset #1, with different energy budgets

hit. For example, the 2HOG+2ACF option only consumes $\approx 54\%$ of the energy consumed by the 4HOG option, while in the former case, 85% of objects actually appeared in the scene were detected, compared to 92% in the latter case. The difference in the achieved accuracy is only $\approx 7\%$.

E. Adaptive choice of algorithms in EECS

Next, we evaluate EECS for adaptively choosing detection algorithms based on the energy budgets and desired detection accuracy. For simplicity, we only show the results for dataset #1 and dataset #2. Similar results are observed in the other dataset.

In this experiment, we choose $\gamma_n = 0.85$ and $\gamma_p = 0.8$, indicating EECS is allowed to reduce the number of cameras, or assign sub-optimal detection algorithms to the cameras, as long as: (i) the number of detected objects is at least 85% of N^*, and (ii) the average detection probability is at least 80% of P^*, where N^* and P^* are the number of detected objects, and the average detection probability of all the detected objects, respectively, when the best algorithms are used by all cameras.

In addition, EECS uses other parameters to control the re-calibration process. In particular, the accuracy assessment period and the re-calibration interval (see section IV-B) are set to 100 and 500 frames, respectively. In other words, EECS uses the detection metadata from 100 frames to assess the detection accuracy and decide the set of cameras and associated detection algorithms. This decision is then used for 500 frames before the accuracy is reassessed again. Note that, for datasets #1 and #2, the ground truth is available every 25 frames. To evaluate EECS, we only process frames that have ground truth information. Thus, EECS actually uses the information from 4 frames to assess the accuracy and select the cameras and detection algorithms. Such selection is then used to process the next 20 frames before re-calibration. In practice, EECS computes the highest possible accuracy (by using the most accurate detection algorithms) using detection metadata. This accuracy measure is then used to calibrate the desired accuracy in place of ground truth information.

Fig. 5 shows the energy consumption and the number of correctly detected humans when (i) the best algorithm is used in each of the 4 cameras, (ii) EECS only chooses a smaller set

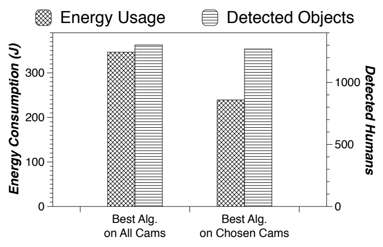


Fig. 6: Detected humans vs. energy consumption for dataset #2

of cameras that is sufficient to achieve the desired accuracy, and finally (iii) EECS chooses sub-optimal algorithms on selected cameras while adhering to the required accuracy.

In Fig. 5a, the energy budget is relatively high so that the camera sensors can choose HOG (the most accurate algorithm) to detect humans. When all cameras use the best algorithm, the whole system consumes ≈ 333 Joules and correctly detects 373 humans in total. However, EECS only needs to use 3 cameras to achieve similar accuracy. If all the *selected* cameras still use the most accurate algorithm (HOG), the energy consumption is reduced to ≈ 248 Joules ($\approx 75\%$ of the highest consumption), while the number of detected humans is 341 ($\approx 91\%$ of the highest accuracy). Further, EECS assigns sub-optimal algorithm (ACF) to some of the cameras to further reduce the energy consumption to ≈ 198 Joules ($\approx 59\%$ of the highest consumption energy cost). This energy conservation is achieved while still detecting 322 humans ($\approx 86\%$ of the highest accuracy).

In Fig. 5b, the available energy budget is less than the energy costs incurred with HOG. Thus, the camera sensors can now only use the sub-optimal algorithm ACF to detect objects. When all the cameras are used, 307 humans are correctly detected and the energy consumption is ≈ 22 Joules. EECS, however, uses fewer cameras (2 or 3 cameras) to achieve similar accuracy. Specifically, the framework can detect 269 ($\approx 88\%$ of the highest accuracy) humans with an energy consumption of only 15 Joules ($\approx 68\%$ of the highest consumption). Since ACF is already the most energy efficient algorithm, EECS cannot further reduce the energy consumption in this case.

In Fig. 6, we show the results for dataset #2. For this dataset, ACF is both the most accurate and most energy efficient algorithm[10]. Thus, the results for this case are similar to those in Fig. 5b. Even though EECS is not able to reduce energy consumption by using more efficient algorithms, it only uses up to 3 cameras (only 2 cameras are used in some rounds) to achieve similar accuracy compared to when all 4 cameras are used. Specifically, EECS is able to correctly detect 1269 humans ($\approx 97\%$ of the highest accuracy), while only consuming 239 Joules ($\approx 70\%$ of the highest consumption).

VII. Discussion

EECS can trade-off the global detection accuracy to some extent (while ensuring a pre-defined accuracy requirement) for energy conservation by assigning sub-optimal algorithms to some of the camera sensors, for certain periods in time. This reduction in detection accuracy could result in a number of undetected objects. However, EECS can be tuned to resist such misses. As it is, objects (e.g., humans) that are not detected in some frames are likely to be detected at other frames (e.g., when the objects move to different locations). In addition, EECS can target energy conservation only in some rounds; thus, a lower detection accuracy is only experienced in such rounds. EECS would then periodically enforce higher accuracy requirements in other rounds to catch objects that were possibly missed earlier. We have performed some preliminary studies that suggest this only results in slightly increased energy costs. Specifically, if HOG and ACF are the most accurate algorithms (as in our data set #1), and HOG yields a higher accuracy with a higher expense, then HOG can be used intermittently to increase accuracy in those corresponding rounds.

[10]The energy consumption values of ACF in Fig. 6 are much higher than in Fig. 5b since the resolution in dataset #2 is significantly higher than in dataset #1.

VIII. Conclusions

In this paper, we present a framework, EECS, for supporting the coordination across a set of camera sensors to achieve a desired object detection accuracy while achieving significant energy savings. Specifically, EECS ensures that cameras do not all unnecessarily use highly accurate but energy heavy video processing algorithms for object detection. In essence, it facilitates the adaptive choice a subset of cameras, and allows some of the chosen cameras use suboptimal detection algorithms to conserve energy while still achieving the pre-defined desired accuracy. Our evaluation shows that EECS helps save more than 40% of the energy consumed compared to a case where all cameras use the highest accuracy algorithms for detection and transfer of key images relating to detected objects. Moreover it still achieves $\approx 86\%$ the accuracy achieved when the best algorithms are used at all of the camera sensors. EECS can be tuned to achieve the right trade-offs between energy efficiency and desired accuracy.

IX. Acknowledgments

This work was partially supported by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF- 09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was also partially supported by the NSF CPS grant 1544969.

References

[1] “CMUcam: Open source programmable embedded color vision sensors.” http://www.cmucam.org.

[2] B. Gong, Y. Shi, F. Sha, and K. Grauman, “Geodesic flow kernel for unsupervised domain adaptation,” in *CVPR*, 2012.

[3] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *CVPR*, 2005.

[4] P. Dollar, R. Appel, S. Belongie, , and P. Perona, “Fast feature pyramids for object detection,” in *PAMI*, 2014.

[5] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part based models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[6] J. Wu, C. Geyer, and J. M. Rehg, “Real-time human detection using contour cues,” in *ICRA*, 2011.

[7] P. Dollar, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *BMVC*, 2009.

[8] R. Gopalan, R. Li, and R. Chellappa, "Domain adaptation for object recognition: An unsupervised approach," in *ICCV*, 2011.

[9] X. Yong, D. Feng, Z. Rongchun, and M. Petrou, "Learning-based algorithm selection for image segmentation," *Pattern Recogn. Lett.*, vol. 26, no. 8, 2005.

[10] O. Mac Aodha, G. J. Brostow, and M. Pollefeys, "Segmenting video into classes of algorithm-suitability," in *CVPR*, 2010.

[11] S. Zhang, Q. Zhu, and A. K. Roy-Chowdhury, "Adaptive algorithm selection, with applications in pedestrian detection," in *ICIP*, 2016.

[12] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *ACM MobiCom*, 2015.

[13] C. Zeng and H. Ma, "Human detection using multi-camera and 3D scene knowledge," in *ICIP*, 2011.

[14] H. Possegger, T. Mauthner, P. M. Roth, and H. Bischof, "Occlusion geodesics for online multi-object tracking," in *CVPR*, 2014.

[15] Y. Li and T. Zhang, "Reducing dram image data access energy consumption in video processing," in *IEEE Transactions on Multimedia*, 2012.

[16] *An Experimental Study on Energy Consumption of Video Encryption for Mobile Handheld Devices*, 2005.

[17] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Integrated power management for video streaming to mobile handheld devices," in *Proceedings of the eleventh ACM international conference on Multimedia*, 2003.

[18] M. Tamai, T. Sun, K. Yasumoto, N. Shibata, and M. Ito, "Energy-aware video streaming with qos control for portable computing devices," in *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, 2004.

[19] L. Zhou, R. Q. Hu, Y. Qian, and H.-H. Chen, "Energy-spectrum efficiency tradeoff for video streaming over mobile ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 5, pp. 981–991, 2013.

[20] B. Schiilkopf, "The kernel trick for distances," *Advances in neural information processing systems*, vol. 13, pp. 301–307, 2001.

[21] J. M. Phillips and S. Venkatasubramanian, "A gentle introduction to the kernel distance," *arXiv preprint arXiv:1103.1625*, 2011.

[22] "iPerf - network bandwidth measurement tool." http://iperf.fr/.

[23] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *CODES/ISSS*, 2010.

[24] "Precision and recall." https://en.wikipedia.org/wiki/Precision_and_recall.

[25] E. Vincent and R. Laganiére, "Detecting planar homographies in an image pair," in *Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis*, 2001.

[26] M. Hirzer, P. M. Roth, M. Koestinger, and H. Bischof, "Relaxed pairwise learned metric for person re-identification," in *ECCV*, 2012.

[27] S. Xiang, F. Nie, and C. Zhang, "Learning a Mahalanobis distance metric for data clustering and classification," in *PR*, 2008.

[28] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *ECCV*, 2006.

[29] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *ICCV*, 2003.

[30] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua, "Multiple Object Tracking using K-Shortest Paths Optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[31] H. Possegger, S. Sternig, T. Mauthner, P. M. Roth, and H. Bischof, "Robust real-time tracking of multiple objects by volumetric mass densities," in *CVPR*, 2013.