Real-Time Multiple Object Visual Tracking for Embedded GPU Systems

Mauro Fernández-Sanjurjo, Manuel Mucientes, and Víctor M. Brea

***Abstract*—Real-time visual object tracking provides every object of interest with a unique identity and a trajectory across video frames. This is a fundamental task of many video analytics applications like traffic monitoring, or video surveillance in general. The development of real-time multiple object tracking systems on low-power edge devices as IoT nodes, without compromising accuracy, is a challenge due to the limited computing capacity of said devices. This might rule out the best in-class computer vision solutions, which nowadays are based on deep learning, and thus, they are very hardware demanding. This paper meets this challenge with a multiple object detection and tracking system that employs cutting-edge deep learning architectures on an embedded GPU while operating in real-time. For this purpose, a system has been designed that extends a joint architecture of tracking and detection by adding a module comprised of appearance-based and movement-based trackers that allow to maintain the identity of the objects of interest for longer periods of time while alleviating the burden of the detector. Our system is mapped onto an embedded GPU platform, cutting down power consumption significantly with respect to a server GPU. Tracking performance metrics show a 51.1% in Multiple Object Tracking Accuracy (MOTA) on the MOT16 dataset. This, in conjunction with a real-time processing speed of 25.2 FPS for up to 45 simultaneous objects and low power consumption of 15W, make our system an ideal solution for a wide-range of video analytics applications.**

***Index Terms*—edge computing, deep learning, multiple object tracking**

I. Introduction

FROM the point of view of the type of information to be transferred to the end user, computer vision applications can be divided into two large groups: (i) those that send a continuous video stream, and (ii) those that only need to send data calculated from video analysis. The latter are especially suitable for edge computing. This is the case of video analytics scenarios, where the computer vision system makes most or all the computing on the edge, before sending the result —which usually does not include video, but data— to the end user [1]–[3]. Some examples of video analytics are: highway tolls, where only vehicle counting and classification in light and heavy classes are of interest; cameras to assess vacancies in a parking lot; closed-circuit TV systems to provide occupancy levels of public events; or video surveillance searching for human anomaly detection like the entrance in restricted areas. This kind of video analytics solutions require lower bandwidths, putting less hardware constraints on the communication network, thus favoring edge computing. Indeed, computing at the edge device makes the system more robust to communication breakdowns and, at the same time, although the uneasiness caused by deployment of cameras in public spaces cannot be assuaged, at least privacy issues from attacks on data can be mitigated —broadcasted data is most of the time processed data instead of raw video.

Computer vision on the edge poses many scientific and technical challenges. Modern computer vision algorithms mainly rely on deep learning, where generally, deeper means more accuracy, but also longer processing times and more underlying computing capacity to support large memory requirements [4], [5]. So, state-of-the-art computer vision deep learning algorithms run on high-end server GPUs, which on many occasions are not an option as edge devices due to their price, size and power consumption, e.g. on-board of a quadcopter. The most straightforward solution is the mapping of algorithms onto embedded GPUs, which feature less computing capacity than that of server GPUs. This usually comes in with time and accuracy penalties from a redesign of accurate but deep models into shorter pipelines to fit on an embedded GPU. The challenge is to design video analytics algorithms on embedded GPUs while keeping accuracy and frame rate.

All the examples of video analytics mentioned above feature object detection and tracking as their main components, providing the objects of interest with an initial bounding box around them and a unique ID along their trajectories. This is a need in most of the video surveillance applications. This is the case of traffic monitoring, sense and avoid on board of UAVs, self-driving cars, or video surveillance assessing of social distance observation [6] —a timely application in COVID-19 times— just to mention a few examples.

Three main components make up these systems: detection, tracking and data association. The detection task consists of identifying the objects of interest in a video frame. Since the introduction of convolutional neural networks for object detection [7], these solutions have been dominating the state-of-the-art. They provide a major improvement in the accuracy obtained and generalize much better throughout different environments. However, their main limitation is computing time and the hardware resources needed for their proper functioning. This is why despite the progress that has been

This research was partially funded by the Spanish Ministry of Science, Innovation and Universities under grants TIN2017-84796-C2-1-R and RTI2018-097088-B-C32, and the Galician Ministry of Education, Culture and Universities under grants ED431C 2018/29, ED431C 2017/69 and accreditation 2016-2019, ED431G/08. These grants are co-funded by the European Regional Development Fund (ERDF/FEDER program).

The authors are with the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain.

made in the field for embedded systems, classic detection solutions are still used in many cases due to the difficulty of achieving real-time processing speed with limited hardware resources using the latest developments of the state-of-the-art.

Once the objects of interest are detected in an image, it is necessary to follow them. This is the task that the tracking module undertakes. Tracking solutions can be differentiated between: appearance-based and motion-based. Appearance-based tracking solutions try, from the initial detection of the object, to find that same object in the successive frames by searching for visual features similar to those previously detected [8]. The two predominant trends in the state-of-the-art in recent years for this type of trackers are those based on discriminative correlation filters [9] and those based on Siamese deep networks [10]. Motion-based trackers try to determine the pattern of the object's movement and, through successive measurements, make a proper prediction of the object's position in the successive frames.

The last part of the system is known as data association, whose mission is to match detections and trackers in order to: associate detections to tracked objects to refine their positions, add new objects that appear in the image, and remove those that have come out. Several methods of data association populate the state-of-the-art, mainly those based on deep networks [11], appearance metrics [12] and distances [13]. In the ideal case of having detections in all the frames of the video, the tracking module would not be necessary and an association between consecutive detections would be enough to solve the problem. However, for a real-time processing system, assuming this detection rate is not realistic without severely compromising the accuracy, especially if the processing is done on an embedded device and cutting edge deep learning architectures are to be used. Therefore, making use of the three parts described above is necessary to provide an optimal solution for this type of hardware.

Designing a detection and tracking solution for an embedded system, which benefits from the advances produced in the field of computer vision in the different modules that form it —detection, tracking and data association— is not a straightforward process. Some of the complexities it presents are:

- Many solutions developed for high end hardware do not work in real-time, therefore, much less will they achieve that desired speed for an embedded system with limited capacity.
- Among those solutions that do work in real-time for a high-end hardware, their computing times increase dramatically when integrated into an embedded system.
- The times reported by most of the algorithms that make up the state-of-the-art of the different parts of the system —detection, tracking or data association— do not take into account the time of the entire system but only of the isolated part under study, something that masks the cost in computation time of the whole system.
- Real-time solutions for embedded systems are commonly formed by modules that work in isolation, usually a lightweight single shot detector and a tracker, without taking advantage of the joint capacity and, therefore, compromising the accuracy of the system.

The work in this paper addresses the problem of multiple object tracking with real-time computing speed and limited hardware resources, while taking advantage of a cutting edge deep learning joint architecture. We design a multiple object tracker system on an embedded GPU, that has much more limited computing capacity than that of high-end server GPUs. This would enable edge computing. Our pipeline comprises a convolutional neural network as object detector to start the tracking of a given object, a visual object tracker, a predictive filter to handle occlusions, and a data association component to deal with multiple objects. We retain many of the advantages in terms of accuracy of a deep learning model designed for high-end hardware yet on an embedded device, while dramatically increasing speed with substantially lower power consumption. These features are very appropriate for their deployment as end device in internet of things applications. The main contributions of this paper are:

1) A complete multiple object tracking system that integrates a CNN detector with embedding information and a tracker. The tracker combines both visual features through a parallel Discriminative Correlation Filter (DCF), and motion estimation with a Kalman Filter.
2) The adaptation of the proposed algorithm for its execution on a low-power embedded system, in this case an NVIDIA Jetson TX2 [14], achieving an optimal balance between speed and precision.
3) The system performs at a rate of 25 FPS on average in the state-of-the-art MOT16 dataset with up to 45 simultaneous objects, thus making the application suitable for real-time processing.

II. Related Work

The first task of a multiple object tracker system is object detection. Until the introduction of convolutional neural networks, two types of object detection dominated the state-of-the-art: object detectors of a specific category and background subtraction algorithms. Object detectors of a specific category were built based on hand-crafted features to provide an image representation, and then they detect objects by searching for those features in the image. Some representatives of these algorithms are Viola & Jones [15], [16], HOG detectors [17] and Deformable Part-based Models (DPM) [18]–[20]. On the other hand, background subtraction algorithms [21]–[23], use several video frames to differentiate the static part of the image (background), and the moving objects that appear in it (foreground). These solutions have the advantage of being fast and not requiring training for specific categories, however, their disadvantages are numerous: they are limited to images taken by static cameras, their accuracy is very dependent on the environment, many artifacts such as shadows can cause problems in detection, and they have difficulties segmenting objects that appear together.

A significant improvement in the accuracy of object detection has been achieved as a result of the introduction of convolutional neural networks. Deep learning object detectors

can be divided into two categories: "two-stage detection" and "one-stage detection". Two-stage detectors use a two-step approach: features extraction and region proposal.

The first paper following this scheme was R-CNN [7], which uses selective search to propose regions where the objects of interest might be, then those regions are fed into a CNN trained in ImageNet [24] to extract features, and finally a linear Support Vector Machine (SVM) classifier determines their presence.

Since this milestone, numerous improvements have been made in the field. Fast RCNN [25] trains a detector and a bounding box regressor under the same network. Then Faster RCNN [26] provided a complete end to end model with the introduction of the Region Proposal Network (RPN). In 2017, Feature Pyramid Network (FPN) [27] was proposed, introducing a top-down architecture with lateral connections for building high-level semantics at all scales. All these architectures might be used in conjunction with different backbones for feature extraction, like: VGG [28], ResNet [5] or ResNeXt [29] among others.

This incremental development of the detector architectures in two phases has allowed to increase considerably the accuracy of the algorithms. However, their computation time is still a great limitation for embedded systems since, first, many of these solutions do not work in real-time and second, those that do it, are only able to do so in high-end hardware. For example, using data from Detectron (Facebook) [30], one of the combinations for detection that offers better results, Faster R-CNN with ResNeXt 101 backbone and FPN, has an inference time of 0.098 sec. This would give an ideal rate of 10.2 FPS, far from real-time. This processing speed is met with an NVDIA V100 GPU with a power consumption of 300 watts compared to 15 watts that would have an embedded system like NVIDIA Jetson TX2. This gives a idea of how far current research is from being fully exploited by low-power embedded systems, which are more suitable for edge IoT devices.

The other type of detectors are 'one-stage' detectors, which directly propose boxes from input images without using a specific region proposal. One of the first approaches of this type was You Only Look Once (YOLO) [31] that divides the image into regions and predicts bounding boxes and probabilities for each region simultaneously. Improvements were made in successive versions [32], especially in YOLO v3 [33] where an FPN-like architecture is used to allow detection of smaller objects. Another algorithm of this type is Single Shot MultiBox Detector (SSD) [34], which proposes a multi-reference and multi-resolution detection techniques. One of the most successful one-stage approaches is RetinaNet [35], that obtains comparable results to those of the two-stage detectors due to its new loss function named focal loss that changes the standard cross-entropy loss, putting more focus on hard misclassified examples during training. Despite being faster than their two-phase counterparts, all these approaches still require more hardware resources than an embedded system like NVIDIA Jetson TX2 to achieve real-time operation [36]. Some lightweight versions of these detectors have emerged such as Tiny-YOLO or MobileNet-SSD [37], [38] which offer much higher processing speed at the cost of lower performance. They are commonly used on embedded architectures for object tracking with real-time speed, which we define in this paper as 25 FPS.

The second part of a multiple object tracker system is object tracking itself. We group the trackers in two categories: visual trackers and motion-based trackers. After an initial detection, visual trackers try to find the object in the successive frames by searching for features similar to those previously detected. One of the solutions of this type of tracking that has had more presence in the field since 2010 [9] have been trackers based on discriminative correlation filters (DCF). They model the appearance of the object of interest through feature filters. Once the position of the object is identified in the first frame, the process continues correlating the filter over a search window in the next frames, and making the new position of the object be the maximum value of the correlation operation map. Initially they were restricted to one feature channel [39], but in the last years major improvements have been made like: multi-channel feature maps [40], scale estimation [41], nonlinear kernels [42], etc. However, with the introduction of deep learning for computer vision, two trends —that use deep features for visual tracking— have emerged to dominate the current state-of-the-art [43]. The first are solutions that exploit the benefits of deep learning with a CNN formulation of the DCF [44]–[46]. The second are based on Siamese networks, which train a deep CNN to solve the similarity learning problem offline, and then evaluate the result online during tracking [10], [47]–[50]. Deep Learning based trackers improve the accuracy of classic algorithms, but they present two major limitations compared with classic DCF solutions with hand-crafted features: many of the latest solutions do not work in real-time [51], [52] and/or their extension from single to multiple objects is not straightforward.

The final part of a multiple object tracker system is data association. It is responsible for integrating the information provided by the detector with the current trackers to: update the position of the trackers with the new detections, add new trackers, and remove those that have disappeared. The advances in this field are represented by those trackers that undertake the Multiple Object Tracking Challenge (MOT). They face the tracking as a data association problem [53], [54] by assuming detections in all frames of a video (i.e. tracking-by-detection). This, as we have mentioned before when discussing object detection, is not a real expectation in many cases, particularly in the case of embedded systems due to their limited hardware resources. Current best performing solutions focus on exploiting the benefits of CNN computing. In [55] they predict the position of an object by exploiting the bounding box regression of an object detector. In [56] they use a joint-inference network to provide an appearance model that enables comparison between trackers, and, thus, increases the robustness of the tracking. These solutions provide promising results but they have a big limitation in terms of computing time, making impossible to achieve real-time processing speed. On the other hand, classic solutions like [13], despite their remarkable speed, do not exploit the features detected in the detection phase, thus creating a lack of essential communication between the detection and tracking processes that hinders

the performance of the system. Some research has emerged that attempts to integrate information from the deep features extracted in the detection network to the data association process to make it more robust [57], [58]. One of the problems they face is that they continue to assume detections in all video frames and/or require high-end hardware to achieve the proposed speed. Both requirements cannot be satisfied on an embedded system.

III. Computer Vision on the Edge

Computer vision on the edge can be run in different modes, i.e., with stand-alone systems on embedded devices [59], cooperatively through edge computing in a wireless sensor network [60]–[62], in IoT distributed networks with edge and cloud computing combined [63], etc. The application at hand will determine the network configuration, as well as the hardware platform and the computer vision algorithms.

Stand-alone multiple object tracker systems typically [1], [57] comprise object detection, one-object tracking, predictive filters like Kalman solutions [64] to handle occlusions, and data association to deal with several objects on the scene [65], [66]. State-of-the-art object detectors and one-object trackers are based on deep learning [67]–[69], and as such demand large memory capacity, along with high-end GPUs not to yield very long detection or tracking times that might render the system useless for video rate processing. Predictive filters and data association, however, are not so demanding, and they can run on CPUs. Still, many core architectures provided with many threads are desirable to parallelize tasks, and track as many objects as possible at a time at video rate processing. In this line, [1], [57] are solutions implemented on server GPUs and many core CPU architectures for tracking several objects at video rate processing. Nevertheless, such a performance cannot be reached on low footprint and power embedded devices, more suitable for edge computing. ASICs and FPGA devices are the best option for embedded computer vision with low footprint and power consumption [70]–[74]. Both, however, suffer from long design cycles and less on-chip programming flexibility, which might make not to take advantage of the latest contributions from the very rapidly evolving field of deep learning for computer vision. This delay between state-of-the-art computer vision solutions and hardware implementations leads to a loss of tracking performance of an embedded approach of this type.

Off-the-shelf microcomputers do not have capacity to hold a stand-alone multiple object tracking system at video frame rate, and they have mainly been used for classification tasks [75]. Thus, embedded GPUs naturally emerge as the best candidate to run our stand-alone multiple object tracker at video rate processing among the plethora of platforms and devices for computer vision on the edge, with the benefits of low footprint and power consumption.

As a result of the previous comments, we have decided to use as embedded hardware for our development the NVIDIA Jetson TX2, which consists of a quad-core 2.0 GHz 64 bit ARMv8 A57 processor, a dual-core 2.0 GHz superscalar ARMv8 Denver processor, and an integrated Pascal GPU

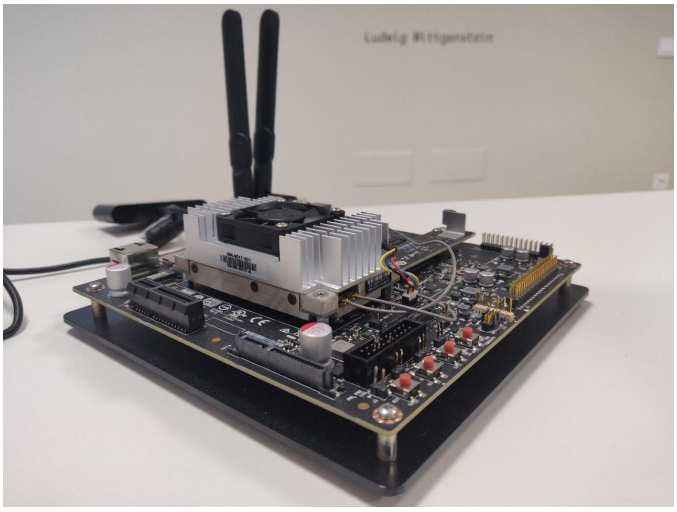


Fig. 1. Jetson TX2 development kit with its integrated GPU was selected as the embedded device for running our system.

1.3 GHz with 256 cores. The six CPU cores and the GPU share 8 GB DRAM memory. We have implemented our multiple object tracking system on the Jetson TX2 development kit which can be seen in Figure 1.

In [59], they also develop a multiple object tracking system on an NVIDIA Jetson TX2 development kit. Their system comprises a background subtraction algorithm for object detection combined with the one-object deep learning tracker GOTURN, instantiated multiple times to deal with several objects on the scene, rendering the tracking of 5 objects at 5 FPS on QVGA video resolution. This multiple instantiation of GOTURN leads to a steep decline in processing speed with the number of objects on the scene. Also, the background subtraction algorithm is not suitable for moving cameras. Our current approach tracks dozens of objects at 25 FPS with a deep learning approach suitable for still and moving cameras.

IV. Multiple object tracking system

The objective of our multiple object tracking system is to identify the objects of interest that appear in a video and track them while they remain on the scene. In case any of these objects becomes occluded the system would try to recover its identity when it reappears. The whole system must reach real-time processing speed. The pipeline of our system is shown in Figure 2. It consists of three main components: detection, tracking and data association. In order to guarantee a real-time processing speed our system will use tracking in all the frames of the video, while detection and data association will be performed only at certain time intervals. This reduces the dependency of our system on the detector, which is the most computationally expensive component. We denote the time between successive detections by τ and, in practice, it is set to the minimum value that allows this real-time speed to be maintained.

When the system receives the first frame of a video, detection is performed and all the detected objects are added as new trackers. The detection (Section IV-B) is done using a

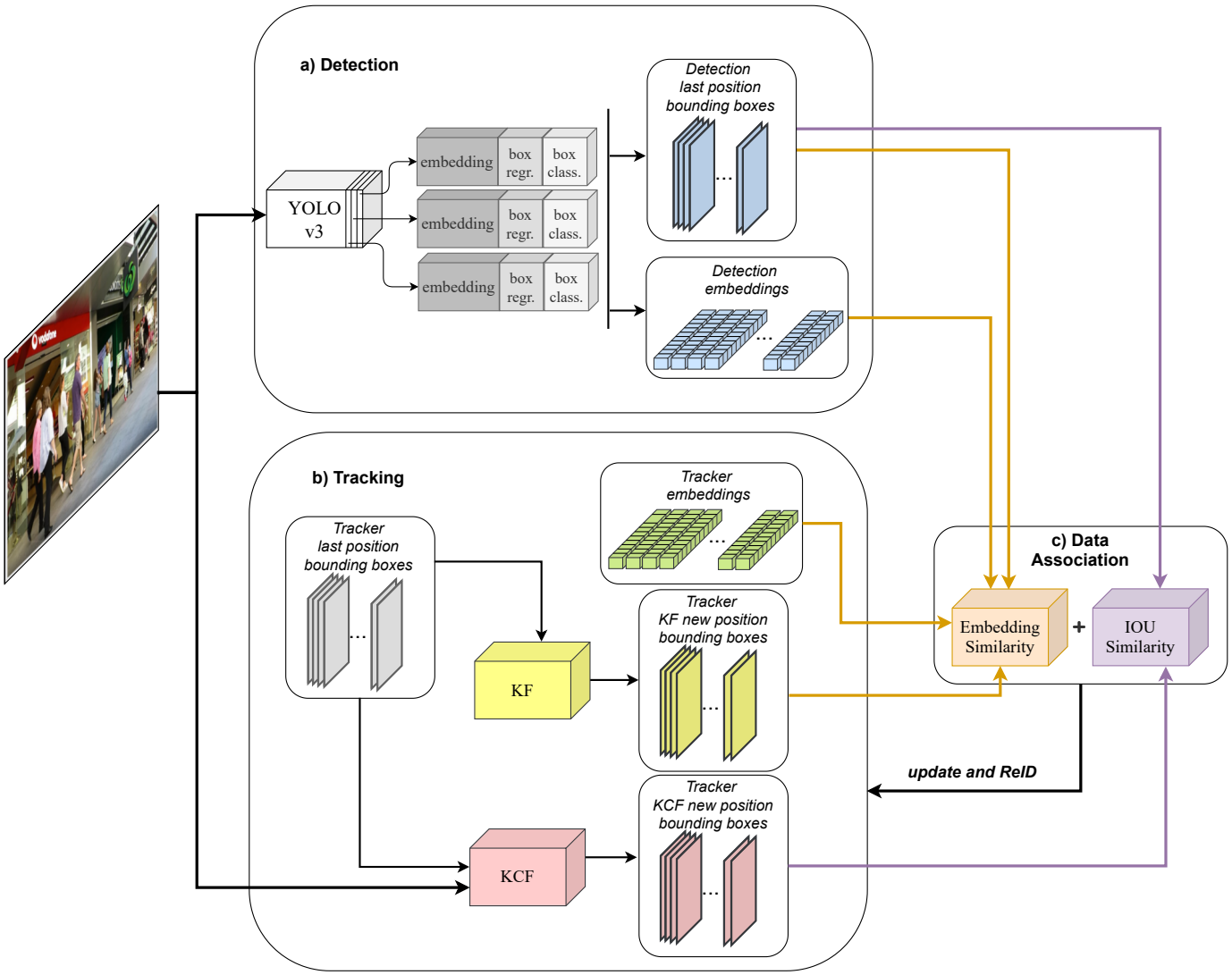


Fig. 2. Different components of the multiple object tracking system. Object detection is carried out by a JDE-based detector —module (a)—, which provides a bounding box for each detected object as well as a feature vector that models its appearance. Tracking —module (b)— is performed by a combination of two different trackers: an appearance-based tracker —Kernelized Correlation Filter (KCF)— and a motion-based Kalman filter (KF). Data association —module (c)— is solved in two steps: (i) associations based on visual features (embedding) which also use motion information; (ii) association by position. Real-time processing speed is achieved through an optimal distribution of the components among the available hardware: detection on GPU, and tracking and data association on CPU.

Joint Detection and Embedding (JDE) based approach [58] —module (a) in Figure 2. This solution combines a deep learning detector model for target localization and an appearance embedding model for data association within the same architecture. Thus, it provides for each detected object a bounding box and an embedding vector that models its appearance.

From this point on, in each new video frame the first task is to estimate the position of the trackers in the new video image through the tracking module —module (b) in Figure 2. This module determines the position of the objects of interest in the successive frames until the next call to the detector, allowing to maintain their identity. The tracking module is composed of a combination of an appearance-based correlation filter tracker —Kernelized Correlation Filter (KCF) [42]— and a motion-based Kalman filter tracker (KF) [64] (Section IV-A). The combination of these two trackers provides increased robustness by making accurate tracking when the object is visible (KCF), and allowing re-identification of occluded objects (KF). Both solutions have the advantage that they are computationally very fast and are fully parallelizable on CPU.

The final process —module (c) in Figure 2— is data association between detections and trackers (Section IV-C). Its main objective is to integrate the information from the other two components to preserve the correct identity of the objects over time. Data association is conducted following two strategies: based on the appearance or the position of the objects. The association based on the appearance of the objects is done through the embeddings provided by the detector, while the one based on position is carried out using the object bounding box. The association is used for: refining the position and features of currently visible objects, adding new objects

that appear in the scene, and removing those that leave the scene or become momentarily occluded. For those frames for which the detector is not run, the system will update the position of the objects using only the information coming from the tracking module.

Algorithm 1: Multiple Object Tracking System

Require:
(a) Im_t: Image frame at current time t
(b) Φ_{t-1}: Set of current visible trackers in last frame $t-1$
(c) Φ_{t-1}^{lost}: Set of lost trackers

1 **Function** `Main(`Im_t, Φ_{t-1}`)`**:**
2 $\quad \Phi_t =$`Track_KCF(`Φ_{t-1}, Im_t`)`
3 $\quad \Phi_t^{lost} =$`Track_KF(`Φ_{t-1}^{lost},Im_t`)`
4 $\quad$ **if** $time\_elapsed > \tau$ **then**
5 $\quad\quad \Psi_t =$`JDE_Detect(`Im_t`)`
6 $\quad\quad \Phi_t^{\alpha^1}, \Phi_t^{\beta^1}, \Psi_t^{\gamma^1} =$`Hung(Emb(`$\Phi_t, \Psi_t$`))`
7 $\quad\quad \Phi_t^{\alpha^2}, \Phi_t^{\beta^2}, \Psi_t^{\gamma^2} =$`Hung(IoU(`$\Phi_t^{\beta^1}, \Psi_t^{\gamma^1}$`))`
8 $\quad\quad \Phi_t^{\alpha^3}, \Phi_t^{\beta^3}, \Psi_t^{\gamma^3} =$`Hung(IoU(`$\Phi_t^{lost}, \Psi_t^{\gamma^2}$`))`
9 $\quad\quad \Phi_t = \{\Phi_t^{\alpha^1}, \Phi_t^{\alpha^2}, \Phi_t^{\alpha^3}, \Psi_t^{\gamma 3}\}$
10 $\quad\quad \Phi_t^{lost} = \{\Phi_t^{\beta^2}, \Phi_t^{\beta^3}\}$
11 $\quad$ **return** Φ_t

Algorithm 1 shows the main steps of the proposed multiple object tracking system. In the following sections, we describe in detail each of these steps.

A. Tracking

The goal of the tracking module is to calculate the position of the previously detected objects in the new video frame. Object tracking is carried out by a combination of two trackers: KCF —appearance-based tracker— and KF —motion-based tracker. On the one hand, the KCF algorithm presents a variant of the classic correlation filter. It identifies the direction in which the object is moving from the maximum correlation between the future patch and any of the translated patches proposed. The model of the object being tracked is updated online using a linear ridge regression model. The main steps of the algorithm KCF can be summarized as:

- With the initial position of the object the model for the tracker is trained in the first frame.
- For a new frame the last bounding box position defines a test image patch.
- The maximum score location over the test image patch and the model represents the new center of the target, and the bounding box is updated with that position.
- The new model is trained at the new position.

This type of tracker has good accuracy and high speed due to exploiting the properties of circular arrays and kernel functions.

On the other hand, Kalman filter-based tracking produces an estimate of the position of the object based on its previous state and the current measurement. The state of the tracked objects is represented by:

$$s = [x, y, a, h, vx, vy, va, vh] \tag{1}$$

where (x, y) represents the bounding box center coordinates, a represents the aspect ratio, h represents the object height and (vx, vy) represents the speeds of the object center shift, and (va, vh) the speeds of the aspect ratio and height change. The KF uses a constant velocity motion model. The state of the object is updated with the measurement (from the matched detection) represented by:

$$z = [x, y, a, h] \tag{2}$$

In the first video frame the trackers —formed by a KCF and a KF— are initialized from the first detections. From that point on, at each time t the inputs to the system are: the current frame of the video (Im_t) and the sets of visible (Φ_{t-1}) and lost trackers (Φ_{t-1}^{lost}) —trackers that could not be matched with previous detections— in the previous frame $t-1$. Each one of the trackers in these sets contains a bounding box that represents its last position, and an embedding vector representing its appearance. The embedding vector is provided by the JDE detector and contains 512-d features to model the appearance of the object. At current time t, the new position of the visible trackers is calculated (Φ_t) in the current frame with the KCF tracker —Function `Track_KCF`, Algorithm 1, line 2 (Alg. 1:2). For the lost trackers the position in the current frame (Φ_t^{lost}) is calculated using the KF tracker —Function `Track_KF`, Alg. 1:3. The purpose of the motion-based tracker is to support the appearance-based tracker when the later fails due to occlusions, distractors or detector failures, thus allowing the eventual re-identification of objects through position. This combination of trackers has the advantages of speed and robustness, making it possible to estimate an accurate position of objects by visual features when they are available, and to predict their movement when they are occluded.

B. Detection

Object detection is done through a JDE approach —module (a) in Figure 2. This solution tackles object detection and embedding modelling within the same architecture, making both parts share the same low-level features and avoiding the re-computation so common in most tracking-by-detection algorithms, which results in a limitation for real-time performance.

The underlying architecture of the detector is a modification of YOLOv3 [33] single shot detector. It has the advantage of being faster than its two-stage detection counterparts and by incorporating an FPN-based architecture [27] for the feature extraction within the network at different scales, which significantly improves the detection of small objects.

To calculate jointly detections and embeddings, once the video image has made a forward pass through the network, a feature map is obtained for each of the three down-sampling scales and these are combined in the way established by FPN. The JDE detection network works as follows: first, the corresponding input frame goes through the network backbone, generating a feature map. In a second step, that feature map enters the header of the network. In a conventional CNN for object detection, the header has two heads: one for the classification of each proposal and another one for refining

the proposal bounding box through regression. However, as seen in Figure 2, the JDE detection adds a third header to generate the embeddings of each detected object.

The frame rate of the system is controlled through parameter τ, which indicates the time elapsed between detections. The challenge is to select the shortest time τ that allows to maintain high frame rate values without affecting tracking metrics with several objects in the scene. In operation, when the time elapsed since the last detection exceeds τ, a JDE-based detection will be performed in the current frame, which will yield a set of object detections (Ψ_t), each with a bounding box and an embedding —Function `JDE_Detect`, Alg. 1:4-5. Once the trackers (Φ_t) and detections (Ψ_t) in the current frame have been calculated, it is necessary to perform data association in order to integrate the information of both sources.

C. Data Association

Data association is done in three steps. First, the assignment between detections and trackers is made using embedding similarity because it provides greater robustness than location-based association. This type of association is based on the appearance of the objects through the use of the deep features provided by the detection network. To carry out the process, a cost matrix is created using the Euclidean distance between trackers and detections embeddings vectors. Given the embedding vectors (em^a, em^b) of size n corresponding to tracker a and detection b respectively, the corresponding matrix element (a, b) —representing row a and column b of the cost matrix C_v— is calculated as:

$$C_v\left(a,b\right) = \sqrt{\sum_{i=1}^{n}\left(em_i^b - em_i^a\right)^2} \tag{3}$$

The value of the matrix C_v contains each possible combination between trackers and detections, that is, the matrix will have as many rows as trackers and as many columns as detections.

To filter out those associations that are not probable, such as two very separate objects, we calculate the Mahalanobis distance. The Mahalanobis distance between a tracker a with mean μ_a and covariance Σ_a —given by its Kalman filter— and a detection b that represents an observation z_b is defined as:

$$C_d\left(a,b\right) = \sqrt{(z_b - j(\mu_a))^T (J\Sigma_a J^T + Q_a)^{-1}(z_b - j(\mu_a))} \tag{4}$$

where $j(\mu_a)$ is the prediction of the measurement for the track with predicted state μ_a, J is the Jacobian matrix of j and Q_a is the covariance of the measurement noise. This is calculated for each possible combination between trackers and detections. For those gating distances that are greater than a threshold η, the cost of association in the matrix ($C_d\left(a,b\right)$) is set to infinity. In practice, this threshold would be the 0.95 quantile of the chi-square distribution with 4 degrees of freedom (9.4877). Finally, the association value of the tracker a and detection b will be a combination of both cost matrices —C_v and C_d—:

$$C_{emb}\left(a,b\right) = \lambda * C_v\left(a,b\right) + (1-\lambda) * C_d\left(a,b\right) \tag{5}$$

The parameter λ determines the trade-off between both costs. In practice, λ is set to 0.98.

Later on the associations are determined by the Hungarian method, which is a well known algorithm for solving the assignment problem in polynomial time, yielding three sets as a result: pairs of detections and trackers successfully associated, represented by a set of current updated trackers $\Phi_t^{\alpha^1}$, unmatched trackers $\Phi_t^{\beta^1}$ and unmatched detections $\Psi_t^{\gamma^1}$ —Alg. 1:6. For each successful assignment $\Phi_t^{\alpha^1}$, the tracker is updated with its corresponding detection in two ways. On the one hand, updating the new bounding box of the tracker with the one of the detection, resetting the KCF and updating the KF in this new position. On the other hand, the feature vector of the last detection (em^b) is used to update the one of the tracker (em^a) by making a weighted sum of them, calculated as:

$$em^a = \delta \times em^a + (1-\delta) \times em^b \tag{6}$$

where the hyper-parameter δ determines the update rate. This way of updating the features allows to progressively refine the embedding vector of the tracker with the new detections.

Unmatched pairings are due to the fact that the minimum similarity threshold between a tracker and a detection is not reached. Those trackers and detections not matched by embedding ($\Phi_t^{\beta 1}, \Psi_t^{\gamma^1}$) will go to the second step of the data association, which is guided by IoU (Intersection over Union) [13]. To make the association by IoU, giving the bounding boxes of tracker a ($bbox^a$) and a detection b ($bbox^b$) the cost is defined as:

$$C_{iou}(a,b) = 1 - \left(\frac{bbox^a \bigcap bbox^b}{bbox^a \bigcup bbox^b}\right) \tag{7}$$

Matrix C_{iou} is again solved by the Hungarian method, which generates the properly updated trackers with their corresponding detections ($\Phi_t^{\alpha^2}$), unmatched trackers ($\Phi_t^{\beta^2}$) and unmatched detections ($\Psi_t^{\gamma^2}$) — Alg. 1:7. In the third step the process is repeated once again with the remaining unmatched detections ($\Psi_t^{\gamma^2}$) and the vector of previously lost trackers (Φ_t^{lost}). This vector contains those trackers that could not be matched in previous time instants, not necessarily only in the previous frame, either due to detector failures or because the object was occluded. This last check allows the re-identification of those objects that reappear on the scene after one of these two situations. The result of this last matching produces the successfully matched trackers ($\Phi_t^{\alpha^3}$), unmatched trackers ($\Phi_t^{\beta^3}$), and unmatched detections ($\Psi_t^{\gamma^3}$) —Alg. 1:8.

Finally, the new set of trackers will be formed by every successfully updated tracker ($\Phi_t^{\alpha^1}, \Phi_t^{\alpha^2}, \Phi_t^{\alpha^3}$) and by those detections that have not been matched up to this point ($\Psi_t^{\gamma^3}$), which will be added as new objects (Alg. 1:9). Those trackers that could not be matched throughout the whole process ($\Phi_t^{\beta^2}, \Phi_t^{\beta^3}$) will form the set of lost trackers (Φ_t^{lost}) —Alg. 1:10. An track remains for a predetermined time t_{delete} in vector Φ_t^{lost} before being completely removed.

By performing the data association in three steps, priority is given to matching based on features learned from the neural network. This implies that in situations where several objects appear together, fewer association errors will be made by

taking into account their appearance first, instead of their position.

V. Experimental Study

In this section, we will present an analysis of the results obtained by our algorithm in different conditions. In Section V-A, we perform a comparison of our system with other three real-time multiple object tracking solutions.. In Section V-B we perform a time analysis discussion about the improvements of our proposal for an embedded architecture.

A. Experimental results and comparison

As quality metrics of our tracking system we will use the two most common in the state-of-the-art: Multiple object tracking precision (MOTP) and Multiple object tracking accuracy (MOTA) [76]. These are calculated between hypothesis (trackers) and objects (ground truth). MOTP reflects the average precision, defined by the average Intersection over Union (IoU) between the hypotheses and the correctly matched objects. MOTA reflects the accuracy of the tracking algorithm by combining three metrics: False Positives (FP), False Negatives (FN) and ID Switches (ID Sw). False positives are those cases in which a hypothesis has no associated object. False negatives occur when an object does not have an associated hypothesis. ID Switches occur when the hypothesis associated with the same object changes. MOTA is the most important of the two metrics and is used to rank the multiple object tracking solutions.

In order to guarantee an accurate measurement of the quality and the execution time of the tracking system, it is necessary to use a widespread dataset with a considerable number of objects. In this case we have selected the MOT16 dataset [54] for several reasons. Firstly, it is the dataset used for JDE evaluation [58], which leads to a fairer comparison with our system. Secondly, it is a standard dataset for measuring MOT metrics and presents a large number of objects in different scenarios. For pedestrian detection we use the model JDE-576×320 provided by JDE authors. We also use the MOT-16 test set for performing the measurements. Figure 3 shows the visual result of our system in some videos of this test set. As usual in a machine learning approach, we have separated training and test sets, so our selected model has been trained with several pedestrian datasets: Caltech, Citypersons, CUHK-SYSU, PRW, ETHZ and MOT17.

To make a fair comparison with [58], it is necessary that the systems to be measured reach a real-time processing speed, i.e. 25 FPS, on the embedded system. The method proposed in [58] works in real-time for a high-end hardware but once integrated on the NVIDIA Jetson TX2 its speed decreases to 3.4 FPS. One of the reasons for this deep fall in processing speed is that the JDE model calls for deep learning detection in every frame. Thus, making straight numbers, video rate processing at 25 FPS with this approach leads to a drop in the number of times the detector is called from once per frame to once every 8 frames on average. That would be the realistic scenario in which a satisfactory processing rate would be reached on the embedded system, we will call this approach JDE-RT.

TABLE I
Tracking results of our system in MOT16 test set.

Sequence	MOTA	MOTP	FP	FN	ID Sw,
MOT16-01	29.2	73.2	562	3,893	72
MOT16-03	67.3	74.7	9,307	23,673	1,184
MOT16-06	25.5	71.5	2,979	5,186	432
MOT16-07	33.4	73.2	2,100	8,459	317
MOT16-08	32.0	74.6	1,244	9,936	194
MOT16-12	43.9	73.7	863	3,663	124
MOT16-14	18.6	69.6	2,525	11,836	684
OVERALL	51.1	74.1	19,580	66,646	3,007

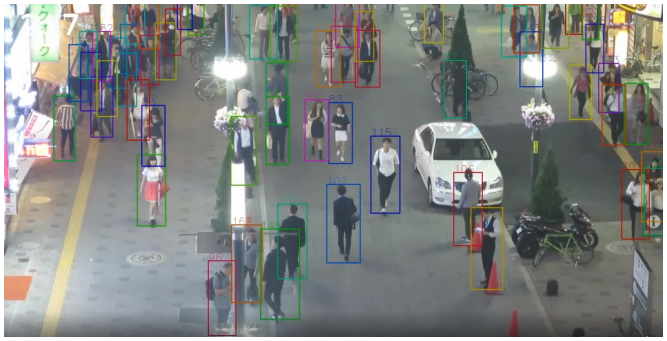
TABLE II
Tracking results of JDE-RT in MOT16 test set.

Sequence	MOTA	MOTP	FP	FN	ID Sw
MOT16-01	25.5	72.9	399	4,327	38
MOT16-03	67.9	74.6	6,709	26,395	466
MOT16-06	8.4	70.4	1,887	8,400	283
MOT16-07	17.9	71.5	1,813	11,363	230
MOT16-08	29.0	73.4	632	11,116	131
MOT16-12	24.9	71.1	727	5,390	115
MOT16-14	1.6	66.9	1,639	15,897	660
OVERALL	45.9	73.9	13,806	82,888	1,923

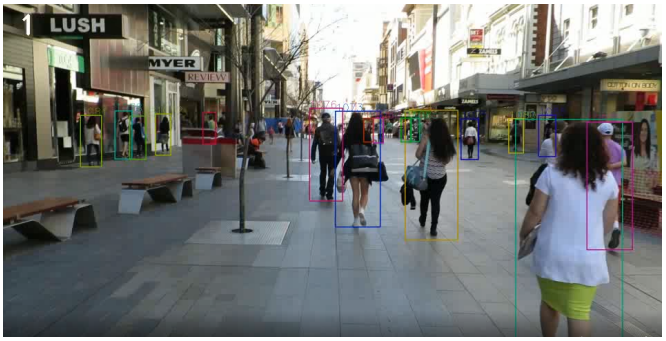
We have also compared our approach with other two real-time solutions. Both of these approaches are based on an implementation of Simple Online Real-Time Tracking (SORT) [77]. This tracker includes a Kalman filter to estimate the position of the objects and the Hungarian method to make the association by Intersection over Union (IOU). For the detection module, these approaches employ Tiny-YOLOv3 [33] and Tiny-YOLOv4, [78] which are the more recent lightweight versions of this single shot detector. These YOLO versions represent the state-of-the-art in real-time object detection on embedded systems. To train both detectors we have used the same dataset as with the other two systems —our approach and JDE-RT. We have called these systems Tiny-YOLOv3-SORT and Tiny-YOLOv4-SORT. Detections have been made in all frames since both approaches work in real-time on Jetson TX2, specifically Tiny-YOLOv3-SORT at 33 FPS and Tiny-YOLOv4-SORT at 30 FPS.

The results of our system (as described in Section IV), which uses detections every 11 frames to achieve the target processing rate, are shown in Table I. Results of JDE-RT —with detections every 8 frames— are shown in Table II. Tables III and IV show the results for the YOLO-SORT based approaches, and finally Table V presents the overall comparison.

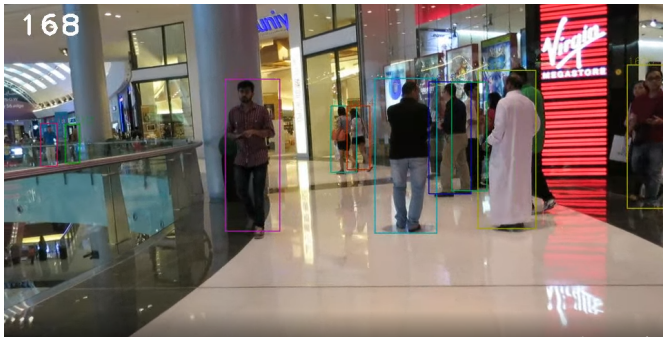
In view of the results we can notice that our system achieves a 5.2 point rise in overall MOTA compared with JDE-RT — or a 11.3% increase. This boost in the accuracy of the system is due to the management of trackers between detector calls. Our solution uses a low-level tracker that adds more robustness than the simple Kalman filter used in the JDE-RT. This implies that the tracked objects will remain overlapped with the ground truth for longer periods of time. Thus, we get an overall better tracking performance by reducing the number



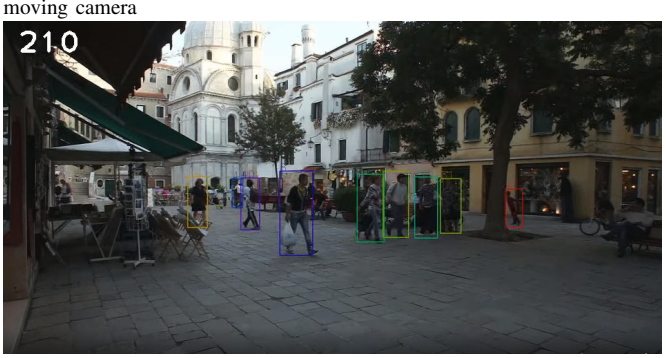
(a) Sequence MOT16-03: Pedestrian street at night, elevated viewpoint



(b) Sequence MOT16-07: A busy pedestrian street filmed at eye level by a moving camera



(c) Sequence MOT16-12: Forward moving camera in a busy shopping mall



(d) Sequence MOT16-01:People walking around a large square.

Fig. 3. Sample frames of our system running on some of the MOT 16 test set videos on the NVIDIA Jetson TX2 embedded device.

TABLE III
TRACKING RESULTS OF TINY-YOLOV4-SORT IN MOT16 TEST SET.

Sequence	MOTA	MOTP	FP	FN	ID Sw
MOT16-01	27.5	79.0	114	4,473	47
MOT16-03	52.9	74.9	3,573	44,893	729
MOT16-06	50.1	78.6	332	5,241	186
MOT16-07	24.6	76.9	334	11,857	119
MOT16-08	32.3	79.2	529	10,645	149
MOT16-12	43.8	78.5	185	4,422	55
MOT16-14	11.6	72.6	225	15,862	260
OVERALL	42.8	75.8	5,292	97,393	1,545

TABLE IV
TRACKING RESULTS OF TINY-YOLOV3-SORT IN MOT16 TEST SET.

Sequence	MOTA	MOTP	FP	FN	ID Sw
MOT16-01	33.9	74.6	139	4,015	71
MOT16-03	51.5	76.5	3,803	46,050	882
MOT16-06	46.5	78.7	300	5,652	219
MOT16-07	26.1	76.4	565	11,333	161
MOT16-08	32.4	80.1	352	10,818	152
MOT16-12	42.2	79.8	125	4,616	57
MOT16-14	13.6	71.4	653	14,943	368
OVERALL	42.3	76.8	5,937	97,427	1,910

TABLE V
COMPARATIVE RESULTS OF ALL SOLUTIONS.

System	MOTA	MOTP	FP	FN	ID Sw
Ours	**51.1**	74.1	19,580	**66,646**	3,007
JDE-RT	45.9	73.9	13,806	82,888	1,923
Tiny-YOLOv4-SORT	42.8	75.8	**5,292**	97,393	**1,545**
Tiny-YOLOv3-SORT	42.3	**76.8**	5,937	97,427	1,910

of false negatives cases (FN) in 16,242.

Both solutions present limitations due to the impossibility of adding a new object or removing one that no longer remains in the scene until the next detector call. Our approach calls the detector less frequently than JDE-RT, which explains the increase in false positives (FP). This lower number of detector calls should have an equally negative impact on the number of false negatives but this is offset by the overall greater robustness of our system. With respect to the number of ID switches (ID Sw), they also increase simply because by being able to track more objects, more associations are performed, and therefore, more errors are made. Regarding multiple object precision (MOTP), our system also improves JDE-RT although the gain is not very significant and both results are high.

Comparing our system with Tiny-YOLOv4-SORT and Tiny-YOLOv3-SORT, we see an improvement in MOTA of 8.3 and 8.8 points, respectively. This is a big difference, especially if we take into account that both approaches use detections in all frames, while our system does it only every 11 frames.

As we can see, the main advantage of our solution is the significant reduction of the number of false negatives (FN), by being able to keep the position of the objects for longer time thanks to the tracking module. YOLO-SORT approaches have fewer false positives (FP) because they perform detections on all frames, thus resetting the bounding boxes of the objects while our system can continue tracking incorrect objects (FP) until the next detection arrives. They also present fewer ID switches (ID Sw) because it is easier to associate objects from consecutive frames than with a separation of 11 frames between them.

Both Tiny-YOLOv4-SORT and Tiny-YOLOv3-SORT out-

perform our system in MOTP by a small margin as both of them always rely on the detector to establish the position of the objects while our system uses tracking most of the time to do so.

As stated before, MOTA gives the same importance to the three errors that negatively impact the tracking of multiple objects —false positives, false negatives or misses, and ID switches. The improvement in MOTA achieved by our solution outperforms the other methods in the comparison (Table V), and makes it an excellent option as a tracking system on embedded low-power devices to solve real-life video surveillance applications. Still, our system features a higher number of false positives than other approaches in Table V. Nevertheless, in a real-life application, this number of false positives could easily be reduced by using knowledge from the application domain at hand, like detecting the areas from where the objects come out, and by removing them at those positions without the need for confirmation from the detector. Overall, the results obtained in the MOT16 test dataset are more than satisfactory given the time and hardware constraints as this dataset is proposed as a real challenge with many objects and occlusions.

B. Time Analysis and Optimization

In this subsection we will give quantitative evidence of the times obtained by the different solutions, as well as a justification of the improvement obtained. First, we compare the complete JDE system as proposed in [58] without any modification using its faster model 576×320, with our system (as explained in Section IV) using the same configuration. These measures have been performed on the NVIDIA Jetson TX2 embedded device and over the same MOT16 test set. Table VI shows the results obtained. The results show that our solution obtains a speed up of 7.4× with respect to JDE. This makes our system suitable for real-time processing.

One of the reasons why it has been possible to achieve this speedup has been the parallelization of the KCF algorithm on the ARM architecture. To illustrate the effect of this optimization, in Figure 4 we show a comparison of this isolated part of the system, that is, only the KCF, comparing the sequential version proposed in [42] against our own optimization on the ARM architecture. This is the bottleneck of our system, and will determine the maximum number of objects that our system is able to manage in real-time. For the sequential KCF implementation, as the number of objects being tracked increases, it can be seen that for more than 12 objects (point A) the system is no longer able to reach real-time processing (25 FPS). However, our optimization, is able to track up to 70 objects (point B) at 25 FPS or more. Thus, our system is able to operate with a large number of objects without sacrificing the application's computing time. The speedup achieved when the workload is maximum (100 objects) is 5.25×. Taking into account that the NVIDIA Jetson TX2 has 6 cores, it is close to the ideal speedup of 6. For the MOT16 dataset the average speed up is 4.03×.

It is worth commenting on the difference in computing capability between high-end GPUs, for which many algorithms are designed, and low-power embedded systems. Table VII shows the comparison between the hardware resources used by the original JDE [58] and our solution. On the one hand, JDE [58] employs an NVIDIA Titan XP GPU, while our system works in real-time on an NVIDIA Jetson TX2 system that has an NVIDIA Pascal GPU. The NVIDIA Jetson TX2 has 8 times less single-precision floating-point performance (FP32) and 16 times less power consumption (Max. PC) than the NVIDIA Titan XP. This gives a rough estimate of the challenge of achieving real time processing speed on hardware with limited resources. That power consumption of 15 W would allow for an energy autonomy of more than 5 hours with a 6.000 mA.h LiPo battery. This allows the execution of the system in environments where it would not be possible if a higher consumption were required.

TABLE VI
AVERAGE FPS ON NVIDIA JETSON TX2

Solution	Avg. FPS MOT16	Speed up
JDE	3.4	–
Ours	25.2	7.4×

TABLE VII
COMPARISON OF HARDWARE RESOURCES BETWEEN NVIDIA TITAN XP GPU USED IN [58] AND NVIDIA JETSON TX2 USED IN OUR APPROACH. THE TABLE SHOWS MAX POWER CONSUMPTION (MAX. PC) AND SINGLE-PRECISION FLOATING-POINT PERFORMANCE (FP32).

Device	Max. PC (W)	FP32 (TFLOPS)
NVIDIA Titan XP GPU	250	12.15
NVIDIA Jetson TX2	15	1.5

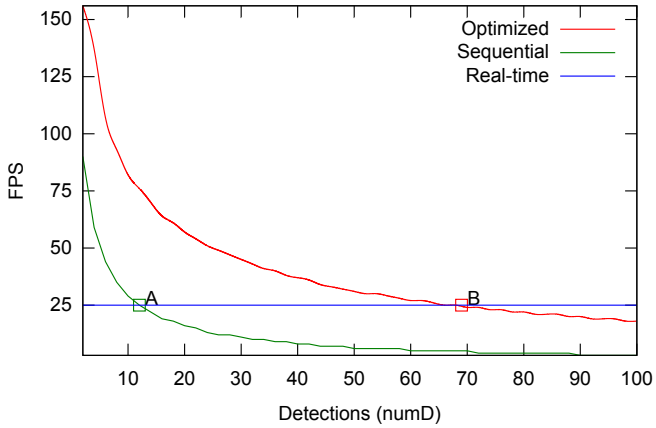


Fig. 4. Comparison of two KCF implementations: sequential (green) and our optimization (red). Real-time speed is represented by the blue horizontal line. The KCF tracker is the bottleneck of our system.

VI. CONCLUSIONS

We have presented a real-time multiple object tracking system for an embedded platform that uses a cutting edge deep learning architecture in conjunction with an additional tracking module —optimized for the embedded architecture— that alleviates the load of the detector and allows for better tracking between widespread detections over time.

The system achieves a real-time processing speed of 25.2 FPS while following up to 45 objects simultaneously. A

comparison of our solution with previous art JDE adapted for real-time processing on an embedded system and two YOLO-SORT solutions has been made using standard multiple object tracking metrics in the MOT16 dataset. Results show an improvement in MOTA ranging from 5.2 to 8.8 points. Also, our parallelization of the low-level KCF tracking algorithm for an ARM architecture has been analyzed, obtaining a maximum speed up of 5.25× with respect to the sequential version.

The system has been deployed on the NVIDIA Jetson TX2 embedded device. Based on the results obtained and taking into account the processing speed and a maximum power consumption of only 15W, our solution is a suitable option for a large number of real-life applications in which video analytics is required.

REFERENCES

[1] M. Fernández-Sanjurjo, B. Bosquet, M. Mucientes, and V. M. Brea, "Real-time visual detection and tracking system for traffic monitoring," *Eng. Appl. Artif. Intell.*, vol. 85, pp. 410–420, 2019.

[2] S. Zhu, C. Chen, and W. Sultani, "Video anomaly detection for smart surveillance," *arXiv preprint arXiv:2004.00222*, 2020.

[3] "UCF-Crime Dataset." [Online]. Available: https://webpages.uncc.edu/cchen62/dataset.html

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016.

[6] "Andrew Ng Web site." [Online]. Available: https://www.andrewng.org/

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2014.

[8] H. T. Nguyen, M. Worring, and R. Van Den Boomgaard, "Occlusion robust adaptive template tracking," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2001.

[9] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2010.

[10] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *Eur. Conf. Comput. Vis. (ECCV)*, 2016.

[11] M. Babaee, A. Athar, and G. Rigoll, "Multiple people tracking using hierarchical deep tracklet re-identification," *arXiv preprint arXiv:1811.04091*, 2018.

[12] W. Choi, "Near-online multi-target tracking with aggregated local flow descriptor," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015.

[13] E. Bochinski, V. Eiselein, and T. Sikora, "High-speed tracking-by-detection without using image information," in *IEEE International Conference on Advanced Video and Signal-based Surveillance (AVSS)*, 2017.

[14] "NVIDIA Jetson. The embedded platform for autonomous everything." [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[15] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2001.

[16] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2004.

[17] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2005.

[18] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2008.

[19] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, "Cascade object detection with deformable part models," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2010.

[20] 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, 2009.

[21] O. Barnich and M. Van Droogenbroeck, "ViBe: A universal background subtraction algorithm for video sequences," *IEEE Trans. Image Process.*, vol. 20, no. 6, pp. 1709–1724, 2010.

[22] M. Hofmann, P. Tiefenbacher, and G. Rigoll, "Background segmentation with feedback: The pixel-based adaptive segmenter," in *IEEE Conference on Computer Vision and Pattern Recognition Workshop*, 2012.

[23] P.-L. St-Charles, G.-A. Bilodeau, and R. Bergevin, "Subsense: A universal change detection method with local adaptive sensitivity," *IEEE Trans. Image Process.*, vol. 24, no. 1, pp. 359–373, 2014.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[25] R. Girshick, "Fast r-cnn," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015.

[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Adv. Neural Inf. Process. Syst. (NIPS)*. Curran Associates, Inc., 2015.

[27] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017.

[30] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016.

[32] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017.

[33] ——, "YOLOv3: An incremental improvement," *arXiv:1804.02767*, 2018.

[34] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Eur. Conf. Comput. Vis. (ECCV)*, 2016.

[35] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017.

[36] S. Hossain and D.-j. Lee, "Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices," *Sensors*, vol. 19, no. 15, p. 3371, 2019.

[37] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[39] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "Exploiting the circulant structure of tracking-by-detection with kernels," in *Eur. Conf. Comput. Vis. (ECCV)*, 2012.

[40] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer, "Adaptive color attributes for real-time visual tracking," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2014.

[41] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Discriminative scale space tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 8, pp. 1561–1575, 2016.

[42] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, 2014.

[43] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, R. Pflugfelder, J.-K. Kamarainen, L. Čehovin Zajc, O. Drbohlav, A. Lukezic, A. Berg, A. Eldesokey, J. Kapyla, and G. Fernandez, "The seventh visual object tracking vot2019 challenge results," 2019.

[44] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg, "ATOM: Accurate tracking by overlap maximization," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019.

[45] G. Bhat, M. Danelljan, L. V. Gool, and R. Timofte, "Learning discriminative model prediction for tracking," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019.

[46] L. Zheng, M. Tang, H. Lu *et al.*, "Learning features with differentiable closed-form solver for tracking," *arXiv preprint arXiv:1906.10414*, 2019.

[47] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[48] Z. Zhu, Q. Wang, B. Li, W. Wu, J. Yan, and W. Hu, "Distractor-aware siamese networks for visual object tracking," in *Eur. Conf. Comput. Vis. (ECCV)*, 2018.

[49] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan, "SiamRPN++: Evolution of siamese visual tracking with very deep networks," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019.

[50] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019.

[51] S. Bai, Z. He, T.-B. Xu, Z. Zhu, Y. Dong, and H. Bai, "Multi-hierarchical independent correlation filters for visual tracking," *arXiv preprint arXiv:1811.10302*, 2018.

[52] T. Xu, Z.-H. Feng, X.-J. Wu, and J. Kittler, "Learning adaptive discriminative correlation filters via temporal consistency preserving spatial feature selection for robust visual object tracking," *IEEE Trans. Image Process.*, vol. 28, no. 11, pp. 5596–5609, 2019.

[53] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015: Towards a benchmark for multi-target tracking," *arXiv:1504.01942 [cs]*, Apr. 2015, arXiv: 1504.01942. [Online]. Available: http://arxiv.org/abs/1504.01942

[54] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A benchmark for multi-object tracking," *arXiv:1603.00831 [cs]*, Mar. 2016, arXiv: 1603.00831. [Online]. Available: http://arxiv.org/abs/1603.00831

[55] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, "Tracking without bells and whistles," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019.

[56] Y.-C. Yoon, D. Y. Kim, K. Yoon, Y.-m. Song, and M. Jeon, "Online multiple pedestrian tracking using deep temporal appearance matching association," *arXiv preprint arXiv:1907.00831*, 2019.

[57] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017.

[58] Z. Wang, L. Zheng, Y. Liu, and S. Wang, "Towards real-time multi-object tracking," *arXiv:1909.12605*, 2019.

[59] B. Blanco-Filgueira, D. García-Lesta, M. Fernández-Sanjurjo, V. M. Brea, and P. López, "Deep learning-based multiple object visual tracking on embedded system for iot and mobile edge computing applications," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5423–5431, 2019.

[60] I. Boulanouar, A. Rachedi, S. Lohier, and G. Roussel, "Energy-aware object tracking algorithm using heterogeneous wireless sensor networks," in *IFIP Wireless Days (WD)*. IEEE, 2011, pp. 1–6.

[61] I. Boulanouar, S. Lohier, A. Rachedi, and G. Roussel, "Pmt 2: a predictive mobile target tracking algorithm in wireless multimedia sensor networks," in *IEEE Symposium on Computers and Communications (ISCC)*, 2014, pp. 1–7.

[62] ——, "Dta: Deployment and tracking algorithm in wireless multimedia sensor networks." *Ad Hoc & Sensor Wireless Networks*, vol. 28, no. 1-2, pp. 115–135, 2015.

[63] C. Neff, M. Mendieta, S. Mohan, M. Baharani, S. Rogers, and H. Tabkhi, "Revamp 2 t: Real-time edge video analytics for multicamera privacy-aware pedestrian tracking," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2591–2602, 2019.

[64] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[65] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[66] B. Wu and R. Nevatia, "Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors," *Int. J. Comput. Vis.*, vol. 75, no. 2, pp. 247–266, 2007.

[67] B. Bosquet, M. Mucientes, and V. M. Brea, "STDnet: exploiting high resolution feature maps for small object detection," *Eng. Appl. Artif. Intell.*, vol. 91, no. 103615, 2020.

[68] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[69] Z. Zhu, Q. Wang, B. Li, W. Wu, J. Yan, and W. Hu, "Distractor-aware siamese networks for visual object tracking," in *Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 101–117.

[70] "Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[71] N. Shah, P. Chaudhari, and K. Varghese, "Runtime Programmable and Memory Bandwidth Optimized FPGA-Based Coprocessor for Deep Convolutional Neural Network," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2018.

[72] M. Rusci, D. Rossi, E. Farella, and L. Benini, "A sub-mw iot-endnode for always-on visual monitoring and smart triggering," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1284–1295, 2017.

[73] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, "Previous: A methodology for prediction of visual inference performance on iot devices," *IEEE Internet of Things Journal*, 2020.

[74] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "Fann-on-mcu: An open-source toolkit for energy-efficient neural network inference at the edge of the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, 2020.

[75] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, "Optimum selection of DNN model and framework for edge inference," *IEEE Access*, vol. 6, pp. 51 680–51 692, 2018.

[76] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[77] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *IEEE International Conference on Image Processing (ICIP)*, 2016.

[78] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

Mauro Fernández-Sanjurjo received the Bachelor's degree in Computer Science in 2014 and Master's Degree in High Performance Computing in 2015, from the University of Santiago the Compostela. Currently, he is a PhD candidate within the Research Center on Intelligent Technologies (CiTIUS) of the University of de Santiago de Compostela (USC). His main research interests are: computer vision, robotics and artificial intelligence.

Manuel Mucientes received his Ph. D. degree in Physics in 2002 at the Universidade de Santiago de Compostela (Spain). Currently he is an Associate Professor of Computer Science and Artificial Intelligence at the Research Center on Intelligent Technologies (CiTIUS) of the Universidade de Santiago de Compostela (USC). His main research interest is artificial intelligence applied to the following areas: computer vision, in the topics of object detection and tracking based on deep learning; machine learning; process mining. He has authored more than 100 scientific papers in these fields of research.

Víctor Manuel Brea received his PhD in Physics in 2003 at the Universidade de Santiago de Compostela (Spain). Currently he is an associate professor within the Research Center on Intelligent Technologies (CiTIUS) of the Universidade de Santiago de Compostela (USC). His main research interests lie in the design of efficient architectures and CMOS solutions for computer vision and micro energy harvesting for IoT applications.