Testbed for Connected Artificial Intelligence using Unmanned Aerial Vehicles and Convolutional Pose Machines

Diego Dantas*, Carnot Braun*, Kaio Forte*, Flávio Brito*, Andrey Silva*, Silvia Lins‡, Neiva Linder†
and Aldebaro Klautau*
*Federal University of Pará, Belém, PA, Brazil
{diego.figueiredo, carnot.filho, kaio.forte, flavio.brito}@itec.ufpa.br, {andreysilva, aldebaro}@ufpa.br
†Ericsson Research, Kista, Sweden
{neiva.linder}@ericsson.com
‡Ericsson Research Brazil, São Paulo, Brazil
{silvia.lins}@ericsson.com

***Abstract*—Unmanned Aerial Vehicles (UAVs) became very popular in a vast number of applications in recent years, especially drones with computer vision functions enabled by on-board cameras and embedded systems. Many of them apply object detection using data collected by the integrated camera. However, several applications of real-time object detection rely on Convolutional Neural Networks (CNNs) which are computationally expensive and processing CNNs on a UAV platform is challenging (due to its limited battery life and limited processing power). To understand the effects of these issues, in this paper we evaluate the constraints and benefits of processing the whole data in the UAV versus in an edge computing device. We apply Convolutional Pose Machines (CPMs) known as OpenPose for the task of articulated pose estimation. We used this information to detect human gestures that are used as input to send commands to control the UAV. The experimental results using a real UAV indicate that the edge processing is more efficient and faster (w.r.t battery consumption and the delay in recognizing the human pose and the command given to the drone) than UAV processing and then could be more suitable for CNNs based applications.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) are becoming a widely used platform with a broad range of sophisticated applications such as data offloading for mobile networks [1], wireless communication [2], hyperspectral remote sensing [3], [4], intelligent transportation systems [5], [6], and disaster monitoring [7], [8]. To develop fully autonomous systems in such applications, is crucial that the drone has an automatic understanding in real time of the visual data collected by the camera [9]. However, a key challenge in the deployment of the object detection task is the limited battery (which affects the flight time) and the computational power to process the visual data in a reasonable time to make critical decisions, such as object avoidance and navigation [10]. Moreover, the task of object detection can face challenges such as image noise and low resolution.

Most of state-of-the-art algorithms for object detection are based on Convolutional Neural Networks (CNNs) such as the works conducted by [11], [12], [13], [14]. Running CNNs locally on UAVs requires a high processing hardware and usually the high power consumption of Graphics Processing Units (GPUs) in (low power) UAVs is prohibitive [10].

The edge/cloud processing and the split of the CNNs through the network [9], [15], [16] is receiving special attention as an alternative to the local processing on UAV. In a scenario with edge/cloud processing without split, the data collected by the UAV is sent to an edge node or to a cloud server for analysis and the result is sent back to the UAV. In the split CNNs scenario, the processing is realized over distributed computing hierarchies (consisting of the cloud, edge and distributed end devices). Both cases enable the UAV to save energy by off-loading computational operations. However, depending on the network conditions and the processing power of the edge/cloud nodes, the wireless transmission of the video can add significant delays to the system. Therefore, for delay-sensitive applications, this kind of solution may not be suitable. Thus, local processing could be recommended [17].

Many UAV studies are being conducted related to the deployment of UAVs with high computational cost tasks due to the increasing necessity of solving complex problems using machines. In [18], the UAV is used for crowd surveillance based on face recognition. In [19], the drones have to carry-out pattern recognition and video preprocessing, both considering the possibility of local processing or offloading the intensive computation tasks to a Mobile Edge Computing (MEC) node. In [9] the authors propose moving the computation to an off-board computing cloud, while keeping low-level object detection and short-term navigation onboard, comparing the Faster R-CNNs [20] algorithm with YOLOv3 [21] and SSD [22]. In the work presented in [16] the authors proposed different scenarios to process the data collected by the drone and a simulation is used to compare two neural networks accuracy: Mask R-CNN [11] and YOLOv3, running exclusively in the drone.

In this work, in contrast, we use OpenPose [23] Neural Network, to detect and classify people's actions, using a drone to capture the video input. The drone uses the OpenPose output

to identify if a person raised his right hand and it takes off if this specific movement is detected. Two scenarios were evaluated for the processing of OpenPose. In the first scenario, the drone processes using the Intel Computer Neural Stick 2 [24]. In the second scenario, video data is sent through the network to a computer which processes the data and returns to the drone the obtained results. In both scenarios, we evaluate the action recognition time and the power consumption of the drone. The results shows that the edge processing is more suitable to this end. However, we highlight that the network conditions are excellent, i.e., no drops or congestion occurs during the experiment. Besides, the edge node is only processing a single UAV application. Therefore, the results can change significantly if more than one UAV is sending data to the same edge or if a background traffic is present.

The main contributions of this paper are listed below.

- We proposed a new algorithm that uses a human pose estimator machine output as its input to detect the human gestures (raised arm) to control the drone (takeoff).
- We successfully developed a proof of concept using a real hardware platform integrated with the power measurement module and an USB VPU that can serve as baseline for future investigations, such as the CNNs network split.
- We evaluate, through extensive field tests, the action recognition time and the battery consumption of the drone running in the two case studies (whole processing locally at the UAV and at the edge) comparing the most appropriated one for this application. Moreover, we evaluate the processing time of each stage when edge processing is being used (network delay, encoding delay, frame extraction and raw processing).

The rest of this work is organized as follows: Section II describes the experiment configuration. Section III provides the evaluation of the obtained results, and Section IV concludes the work.

II. Testbed Description

The testbed consists of using the OpenPose to extract the human pose from 2D images and detect certain human movements to control a drone. The action recognition time is calculated based on the total time needed for one movement to be detected. Everything runs on a real drone but the motors were turned off so the drone doesn't fly and only receives commands as if it were flying. In the following, we describe in details all the experiment parameters.

A. Use cases

We developed the testbed to use the drone for comparing action recognition time and battery consumption for the use cases presented in Fig. 1.

Both scenarios are described as follow:

- **CNNs running exclusively in the drone (Fig. 1, scenario 1):** Applications using CNNs are computationally demanding which makes their use in low cost UAVs infeasible. However, nowadays there are already small VPUs that can be easily carried by drones, such as the Intel Neural Compute Stick 2 [24], which is a very competitive solution available on the market, where the whole data processing can happen in real-time on companion computers with low power processing capabilities using a USB port.
- **CNNs running in the edge (Fig. 1, scenario 2):** In this scenario, the drone send the data directly to the edge. The data processing is performed, and the output is sent back to the drone. Under those circumstances, the computational cost for UAVs is very low. The network conditions will be ideal, which means that network delay time evaluation can not be generalized but still giving reasonable estimation.

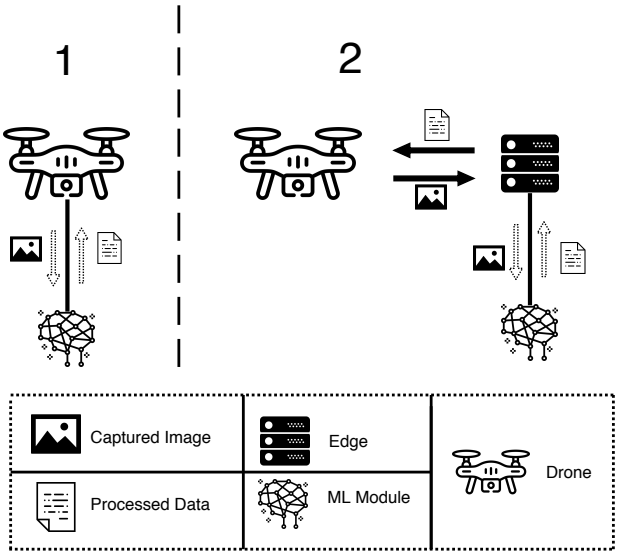

Fig. 1: User cases scenarios.

Table I describes the configuration of each scenario developed.

	Scenario 1	Scenario 2
CPU	Intel Atom x7-Z8750 @ 2.56 GHz	Intel Core i7-7700 @ 3.60 GHz x 8
GPU	None	Nvidia GeForce RTX 2070
VPU	Intel Movidius Myriad X	None
RAM	4 GB	32 GB

TABLE I: Configuration for the two scenarios.

B. Drone details

We used the drone known as Intel Aero Drone [25], which is flexible and is able to connect a wide variety of sensors and peripherals. The drone can communicate with other devices such as smartphones or laptops over a standard WiFi network, which gives to it the ability to transfer commands via another device. The camera resolution is 640x480 pixels, with each frame having 45Kb of size.

C. Setup for power measuring

To obtains the battery consumption estimation, we developed a small circuit on a breadboard (since the drone is

staying on the ground) using an Arduino Mega 2560 [26], a voltage divider and the ACS712 [27] current sensor module connected (in parallel and series, respectively) with the battery. The voltage divider consists of 470KΩ and a 120KΩ resistors with 5% accuracy for solid measurement. The ACS712 uses the Hall effect to transform the current into a voltage value that can be read by the Arduino.

Both sensors had their outputs connected to the analog input ports of the microcontroller, and then the voltage and current values are measured making the appropriate conversion using the constants related to each sensor. Finally, these measured values were transmitted through the Arduino serial port connected to a computer running Linux, where they were saved for further analysis. Fig. 2 shows the equipment used for simulations.

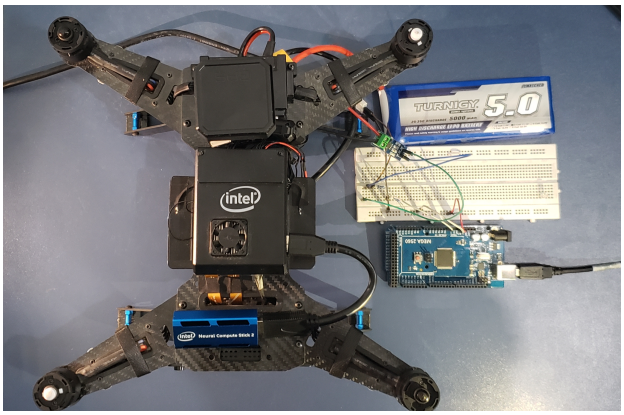

Fig. 2: Equipment used in the testbed.

D. Movement detector

The movement selected for the experiment is one person raising its right arm. After detecting the movement, the drone should takeoff. Fig. 3 shows an illustration of the expected drone reaction after the human command.

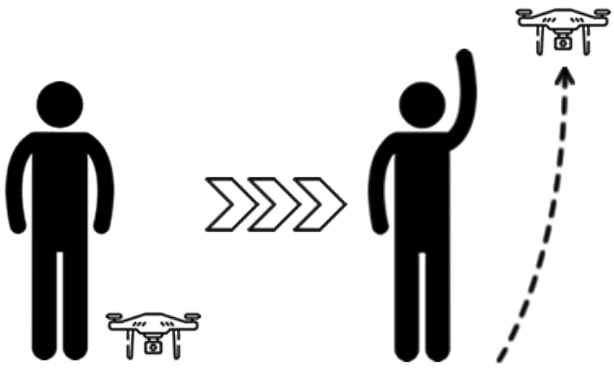

Fig. 3: Experiment action controlling the drone.

The algorithm to detect the movement is based in the OpenPose output. First, it predicts the coordinate points where the arm would be when the arm is raising. To this end, the knowledge of the coordinates of the head and the right leg is necessary. Every point will follow a trajectory up and be within a specific region that is possible to be characterized as shown in Fig. 4.

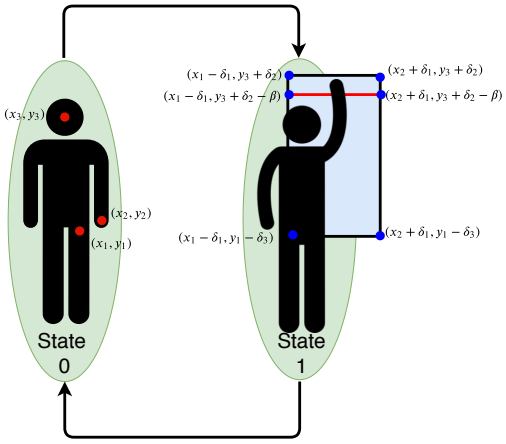

Fig. 4: Points of interest for the algorithm.

Algorithm 1 Main procedure for movement detector

```
state ← 0
procedure PROCESS POSE(HumanPose)
    if state = 0 then                          ▷ Waiting for initial position
        state ← initial_position(HumanPose)
        if state = 1 then                      ▷ Initial position verified
            calculate_box(HumanPose)
        end if
    else                                       ▷ Action is being performed
        state ← trajectory_check(HumanPose)
        if state = 2 then                      ▷ Action was performed
            state ← 0                          ▷ Back to initial state
            return true
        end if
    end if
    return false                               ▷ Action was not performed
end procedure
```

The *State* 0 is the initial state, and it waits for the person to be standing straight with the right arm down. This condition can be verified by checking two different euclidean distances. The first one is between the point located in the right thigh and the point located in the right hand. In Fig. 4, the right thigh coordinates are defined as (x_1, y_1) and the right hand as (x_2, y_2). Then, the euclidean distance α_1 can be found using Equation 1.

$$\alpha_1 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

The second euclidean distance necessary is between the head point and right hand point. In Fig 4, these coordinates are (x_3, y_3) and (x_2, y_2) respectively and the distance α_2 can be found using Equation 2.

$$\alpha_2 = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} \tag{2}$$

After calculate the distances α_1 and α_2, the system determines if the right arm is down by checking the conditions

$\alpha_1 < \beta_1$ and $\alpha_2 < \beta_2$, where β_1 and β_2 are constant values. If both conditions are satisfied, then the system goes from *State* 0 to *State* 1.

In *State* 1, the first step is to design the blue rectangle and Fig 4 shows the coordinated of the rectangle design. Basically, we use the three points (x_1, y_1), (x_2, y_2) and (x_3, y_3) founded in *State 0* and more three variables named δ_1, δ_2 and δ_3 which is calculated using Equations 3, 4 and 5. The fourth variable is β which is a constant value. Using all these variables, the rectangle can be created as Fig 4 illustrates.

$$\delta_1 = \frac{|x_1 - x_2|}{2} \tag{3}$$

$$\delta_2 = \frac{|y_1 - y_3|}{2} \tag{4}$$

$$\delta_3 = \frac{|y_1 - y_3|}{4} \tag{5}$$

After the rectangle creation, the current right-hand position (x_2, y_2) is stored and a loop starts. At each loop iteration, the system receives the current right hand position (x_{2c}, y_{2c}). With the new current position (x_{2c}, y_{2c}) and the last position (x_2, y_2), three observations are performed for checking the completion (or not) of the movement. The first observation is to verify if the current point (x_{2c}, y_{2c}) is out of rectangle delimited by the points. The second is to certify if the current point (x_{2c}, y_{2c}) is within the rectangle region, and if the current point y_{2c} is higher than y_2. Then, the last observation is to verify if the person's hand is within the interest region (rectangle region), if positive, the algorithm successfully detected the movement. The complete operation the proposed action recognition scheme is detailed in Algorithm 1, where the functions *calculate_box* and *trajectory_check*, are responsible for the rectangle design and the verification of the right-hand trajectory. The algorithm also needs to ensure that the person is standing and not in another position (such as lying down), and it can be done by ensuring that the x coordinates of the head and thigh are closer.

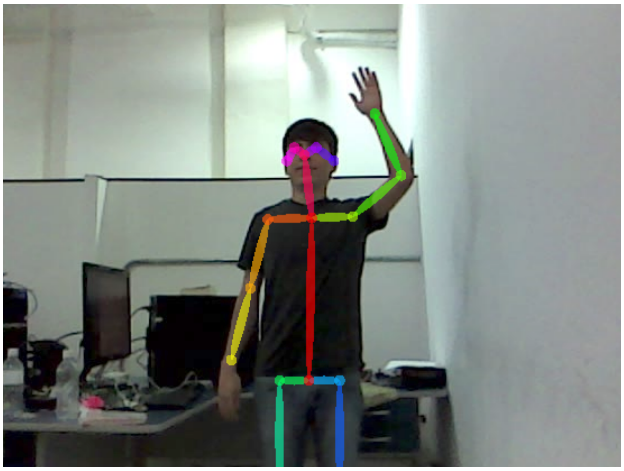

Fig. 5: A scene of the video processed by OpenPose used in the experiment.

III. Results

In this section, we evaluate the action recognition time and the battery consumption performance of the drone application in both scenarios proposed for the testbed. All experiments were conducted in an indoor environment. For a fair comparison of the battery usage, the energy consumption was evaluated until the drone received the takeoff signal thus not considering the power consumption from motors. So we can consider that the achieved result represents the use of one battery exclusively for the companion computer.

The camera video is passed as input to the algorithm. Matrix A, B, and the Equation 6 show the acquisition process of the results for both power consumption and recognition time. Each element $a_{i,j}$ of the matrix A represent the time, or power consumption, needed for one action be recognized and therefore we will call it one iteration. Each row in matrix A represents a sample composed of successive 50 iterations, that is, the battery of the drone was not recharged affecting the power consumption analysis. We take the mean of each column of matrix A with Equation 6 representing the average value for both parameters being analyzed at a given level of the battery, and matrix B is generated with these values.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,50} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,50} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,50} \\ a_{4,1} & a_{4,2} & a_{4,3} & \dots & a_{4,50} \\ a_{5,1} & a_{5,2} & a_{5,3} & \dots & a_{5,50} \end{bmatrix}$$

$$B = \begin{bmatrix} f(1) & f(2) & f(3) & \dots & f(50) \end{bmatrix}$$

$$f(j) = \sum_{i=1}^{5} a_{ij} \tag{6}$$

The action recognition time may seem impractical in both use cases for real time systems but this delay is expected given the total time necessary for human actions to be performed. In general, is expected that the delay of local processing would be lower than edge or cloud processing, due to the network delay that is added to the process. In opposite, the results illustrated in Fig. 6 shows that the action recognition time at the edge is approximately 50% lower than processing locally. This occurs due to the lower processing power of the drone (even using the VPU). The VPU is not able to process the CNN as fast as the GPU at the edge. Besides, the network conditions are ideal, e.g., there is no background traffic, which contributes to the reduction of the delay using edge processing. More realistic scenarios involving heterogeneous background traffic will be evaluated in a future work.

Besides the action recognition time at the edge, we evaluated the processing time of each stage involving the edge to determine where the delay is more critical. Analyzing Fig. 7, the network delay [28] (i.e., the time to send and receive the data) is the most impacting, having an average about 2.2 seconds, approximately. The encoding delay, and the edge processing are about 1 second, and the frame extraction (from the camera video) delay is about 0.3 seconds. These information can be useful to know where to investigate methods to

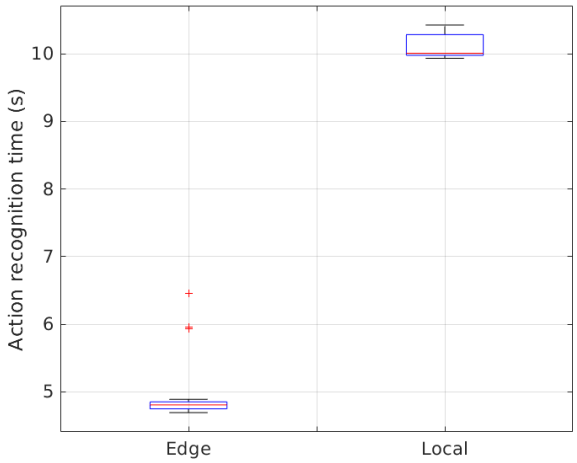

Fig. 6: Action recognition time for each approach.

optimize the edge system. In other words, since the network delay is responsible for almost 50% of the delay to recognize the action, developing methods to improve this communication will have more relevance for delay mitigation.

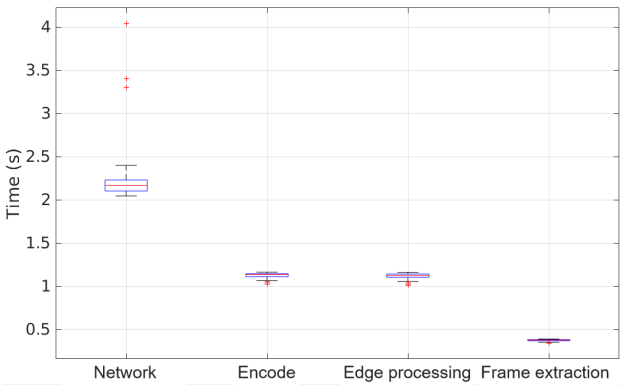

Fig. 7: Edge processing time at different stages.

Reduced power consumption is expected in the edge processing scenario due to the reduced number of operations performed in the UAV. The difference between both approaches is approximately 1 W, as shown in Fig. 8. The local processing has an average power consumption about 11.43 W, and the edge processing is about 10.47 W.

Equations 7 and 8 show the estimated battery lifetime for the local processing and edge processing, respectively.

$$\frac{55.5\,Wh}{11.43\,W} = 4.86\,h \tag{7}$$

$$\frac{55.5\,Wh}{10.47\,W} = 5.30\,h \tag{8}$$

The battery lifetime using the edge is about 9% greater than local processing. Therefore, edge processing can be more suitable for scenarios where network conditions are excellent and in situations where the battery lifetime and reaction time are critical requirements.

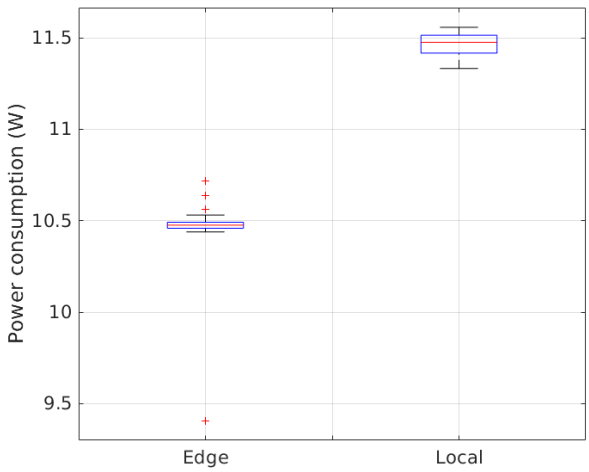

Fig. 8: Power consumption of scenarios for each approach.

It is important to notice that the given values of battery lifetime are not considering the rotors of the drone, therefore, in reality, they are smaller.

IV. CONCLUSIONS

Real-time object detection relying on CNNs is computationally demanding and process it on a low-cost UAV platform is a challenging task. In this paper, we evaluated the impacts of processing CNN at UAV and in the edge, using an application running on top of OpenPose to detect the human gesture to control the drone. We verified through the results that edge processing is more efficient and faster than local processing when the network conditions are favorable. Moreover, we successfully developed a testbed that will drive our next work to evaluate more challenging scenarios, such as the split of CNN through the network. For future works, we intend to use more realistic scenarios, such as different levels of background traffic, increased number of UAVs (through emulation), and different kind of applications in the drone using different CNNs architectures. We also will compare the proposed algorithm to recognize if a person raised its arm with other solutions to prove its viability.

ACKNOWLEDGMENT

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A, CNPq and Capes Foundation, Brazil.

REFERENCES

[1] Fen Cheng, Shun Zhang, Zan Li, Yunfei Chen, Nan Zhao, F Richard Yu, and Victor CM Leung. Uav trajectory optimization for data offloading at the edge of multiple cells. *IEEE Transactions on Vehicular Technology*, 67(7):6732–6736, 2018.

[2] Qingqing Wu, Yong Zeng, and Rui Zhang. Joint trajectory and communication design for multi-uav enabled wireless networks. *IEEE Transactions on Wireless Communications*, 17(3):2109–2121, 2018.

[3] Helge Aasen, Eija Honkavaara, Arko Lucieer, and Pablo Zarco-Tejada. Quantitative remote sensing at ultra-high resolution with uav spectroscopy: a review of sensor technology, measurement procedures, and data correction workflows. *Remote Sensing*, 10(7):1091, 2018.

[4] Yanfei Zhong, Xinyu Wang, Yao Xu, Shaoyu Wang, Tianyi Jia, Xin Hu, Ji Zhao, Lifei Wei, and Liangpei Zhang. Mini-uav-borne hyperspectral remote sensing: From observation and processing to applications. *IEEE Geoscience and Remote Sensing Magazine*, 6(4):46–62, 2018.

[5] Mariem Maiouak and Tarik Taleb. Dynamic maps for automated driving and uav geofencing. *IEEE Wireless Communications*, 26(4):54–59, 2019.

[6] Vishal Sharma, Ilsun You, Giovanni Pau, Mario Collotta, Jae Lim, and Jeong Kim. Lorawan-based energy-efficient surveillance by drones for intelligent transportation systems. *Energies*, 11(3):573, 2018.

[7] Akimitsu Kanzaki and Hideyuki Akagi. A uav-collaborative sensing method for efficient monitoring of disaster sites. In *International Conference on Advanced Information Networking and Applications*, pages 775–786. Springer, 2019.

[8] Yuan Man, Tianjie Lei, Xuemei Liu, Shican Li, Xiangyu Li, Jintao Huang, and Jiabao Wang. The application of uav remote sensing in natural disasters emergency monitoring and assessment. In *Eleventh International Conference on Digital Image Processing (ICDIP 2019)*, volume 11179, page 1117919. International Society for Optics and Photonics, 2019.

[9] Jangwon Lee, Jingya Wang, David Crandall, Selma Šabanović, and Geoffrey Fox. Real-time, cloud-based object detection for unmanned aerial vehicles. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 36–43. IEEE, 2017.

[10] George Plastiras, Christos Kyrkou, and Theocharis Theocharides. Efficient convnet-based object detection for unmanned aerial vehicles by selective tile processing. In *Proceedings of the 12th International Conference on Distributed Smart Cameras*, page 3. ACM, 2018.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[12] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[13] Jonathan Pedoeem and Rachel Huang. YOLO-LITE: a real-time object detection algorithm optimized for non-gpu computers. *arXiv preprint arXiv:1811.05588*, 2018.

[14] Pengyi Zhang, Yunxin Zhong, and Xiaoqiong Li. SlimYOLOv3: Narrower, faster and better for real-time uav applications. *arXiv preprint arXiv:1907.11093*, 2019.

[15] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339. IEEE, 2017.

[16] Flavio Mendes, Andrey Silva, Aldebaro Klautau, Neiva Linder, Silvia Lins, Kaio Henrique, Carnot Braun, and Diego Dantas. Testbed development and evaluation of drone-based real-time object detection using deep learning. *LANCOMM*, 2019.

[17] Christos Kyrkou, George Plastiras, Theocharis Theocharides, Stylianos I Venieris, and Christos-Savvas Bouganis. Dronet: Efficient convolutional neural network detector for real-time uav applications. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 967–972. IEEE, 2018.

[18] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. Uav-based iot platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134, 2017.

[19] Mohamed-Ayoub Messous, Sidi-Mohammed Senouci, Hichem Sedjelmaci, and Soumaya Cherkaoui. A game theory based efficient computation offloading in an uav network. *IEEE Transactions on Vehicular Technology*, 68(5):4964–4974, 2019.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[23] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008*, 2018.

[24] A plug and play development kit for ai inferencing. https://software.intel.com/en-us/neural-compute-stick, 2019. Last accessed 09 September 2019.

[25] Intel aero ready to fly drone is a ready-to-fly unmanned aerial vehicle (uav) development platform. https://click.intel.com/intel-aero-ready-to-fly-drone-2730.html, 2019. Last accessed 09 September 2019.

[26] The arduino mega 2560 is a microcontroller board based on the atmega2560. https://store.arduino.cc/usa/mega-2560-r3, 2019. Last accessed 09 September 2019.

[27] Fully integrated, hall effect-based linear current sensor with 2.1 kvrms voltage isolation and a low-resistance current conductor. https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf, 2019. Last accessed 09 September 2019.

[28] Mario Baldi and Yoram Ofek. End-to-end delay analysis of videoconferencing over packet-switched networks. *IEEE/ACM Transactions On Networking*, 8(4):479–492, 2000.