2020 IEEE 7th International Workshop on Metrology for AeroSpace (MetroAeroSpace)
22-24 June 2020, Pisa, Italy. DOI:10.1109/MetroAeroSpace48742.2020.9160214

Adaptive Detection Tracking System for Autonomous UAV Maritime Patrolling

Alessandro Panico
Flight Test Wing
Italian Air Force
Pratica di Mare, ITA
panicoale@gmail.com

Luca Zanotti Fragonara
Centre for Autonomous and
Cyber-Physical Systems
Cranfield University, UK
l.zanottifragonara@cranfield.ac.uk

Saba Al-Rubaye
Centre for Autonomous and
Cyber-Physical Systems
Cranfield University, UK
s.alrubaye@cranfield.ac.uk

***Abstract*—Nowadays, Unmanned Aerial Vehicles (UAVs) are considered reliable autonomous systems suitable for several autonomous applications, especially for target detection and tracking. Although significant developments were achieved in object detection systems over the last decades using the deep learning technique known as Convolutional Neural Networks (CNN), there are still research gaps in this area. In this paper, we present a new object detection tracking algorithm that can apply on low power consuming processing boards. In particular, we analysed a specific application scenario in which a UAV patrols coastlines and autonomously classifies different kind of marine objects. Current state of the art solutions propose centralised architectures or flying systems with human in the loop, making the whole system poorly efficient and not scalable. On the contrary, applying Artificial Intelligence (AI) detection system that runs on commercial Graphics Processing Units (GPUs) makes UAVs potentially more efficient than humans (especially for dull tasks like coastline patrolling) and the whole system becomes easily scalable because each UAV can fly independently and the Ground Control Station does not represent a bottleneck. To deal with this task, a database consisting of more than 115.000 images was created to train and test several CNN architectures. Furthermore, an adaptive detection tracking system was introduced to make the whole system faster by optimizing the balance between new detections and tracking existing targets. The proposed solution is based on the measure of the tracking confidence and the frame similarity, by means of the Structural SIMilarity (SSIM) index, computed both globally and locally. Finally, the developed algorithms were tested on a realistic scenario by means of a UAV test-bed.**

***Index Terms*—UAV, CNN, Detection-Tracking System**

LIST OF ACRONYMS

ANN	Artificial Neural Network
CSRT	Channel and Spatial Reliability Tracker - Discriminative Correlation Filter
CNN	Convolutional Neural Network
FPS	Frames Per Second
GPU	Graphics Processing Unit
HSV	Hue, Saturation, Value
IoU	Intersection over Union
KCF	Kernelized Correlation Filter
mAP	Mean Average Precision
ML	Machine Learning
R-CNN	Region-based Convolutional Neural Network
RoI	Region of Interest
ROS	Robot Operative System
RPN	Region Proposal Network
FPN	Feature Pyramid Network
SPP	Spatial Pyramid Pooling
SSD	Single Shot Detector
SSIM	Structural SIMilarity
SVM	Support Vector Machine
SW	Software
UAV	Unmanned Aerial Vehicle
VGG	Visual Geometry Group
YOLO	You Only Look Once

I. INTRODUCTION

The localization of sharks and other marine targets along coastlines is a valuable service in those countries in which a significant number of shark attacks was reported over the last decades. This phenomenon is mainly related to the population growth and the increasing number of human activities in the oceans, having severe social and economic implications [1]. Conventionally, the approaches to face this problem are based on marine deterrents (nets, drum-lines) and spotters (submarine or aerial). The latter method is typically preferred due to the lower impact on the marine ecosystem. Thus, this paper aims to design a shark classifier (a spotter) by training a Artificial Neural Network (ANN) to be executed on a low performance Graphics Processing Unit (GPU) embarked on a Unmanned Aerial Vehicle (UAV) that patrols coastlines and sends alerts to the lifeguards control station if sharks are detected. Similarly, the same classifier might be used to face collateral marine problems like mammals preservations, human activity and pollution monitoring. This paper is organized as follows. Firstly, an overview of the state of the art solutions for aerial detection of marine targets (Section II). After that, the project methodology is described in Section III, showing how the database was created, the training of the Convolutional Neural Network (CNN), the implementation of the adaptive detection tracking algorithm and its integration with a UAV test-bed. Finally, the analysis of the project results is presented in Section IV, whereas the final remarks and the future work proposals conclude the manuscript.

II. LITERATURE REVIEW

Object detectors can be essentially designed by means of two different approaches. The classic programming schema, based on image processing techniques, suggests to classify the objects by using a flowchart with subsequent if-then rules. On the contrary, innovative machine learning methods are based on autonomous feature extraction, in which the computer derives the specific characteristics of each class of targets and learns how to recognize them in new images. The autonomous marine wildlife classification problem was initially approached by [2], with the specific aim of detecting dugongs by the use of imaging systems at an altitude of 1000 feet. Their algorithm performed a red-ratio threshold in the Hue, Saturation, Value (HSV) domain to identify possible candidates and morphological filters to remove bright spots. Final decision on the object class was carried out by an algorithm that evaluated the object elongation, relating highly elliptical shapes to higher probability of facing a dugong. Similarly, in [3] the flowchart is improved by an adaptive red-ratio threshold in the HSV domain, allowing slight improvements in the detection accuracy, whereas the reduction of false positives was addressed by the introduction of multi-layer filters in [4]. However, the detections were not satisfactory in all cases. The exploitation of multi-spectral products was proposed by [5], in order to detect and roughly classify different shark species making use of classical techniques and achieving a classification precision of 84% with 66 observations, giving promising results even with different submerging conditions. The second part of this section focuses on Machine Learning (ML) techniques. The simultaneous application of image processing and ML was investigated by [6], in which relevant candidates selected with conventional techniques were identified by means of a trained ANN classifier, achieving a Mean Average Precision (mAP) of 90%. However, the results of this study still depend on the capacity to identify objects and extract their relevant features. The introduction of the CNN allowed to by-pass the initial anomalies detection algorithms, as shown in [7]. These networks were developed to analyse data that were positionally related (e.g. in the space/time/frequency domain). Due to this technique, the feature extraction process becomes an intrinsic task in the inner layers of the neural network. The goal of the research was to develop an efficient tool for the automated and real-time coastline monitoring, specifically tuned for shark identification. In order to carry out this task, a database of almost four thousands video frames was used for the training, validation and testing of a Faster Region-based Convolutional Neural Network (R-CNN), achieving an average precision of 0.904, distinguishing among four classes. The inner network architecture that produced these remarkable results was the Visual Geometry Group (VGG)-16, introduced by [8]. This network is made of 13 convolutional layers, followed by 3 fully connected layers. However, the project bottleneck was the processing time: about 7 FPS on high performance HW, being suitable to be executed on the ground segment only. Therefore, the UAVs were only responsible to acquire and transmit the video to the ground station. The same research led to a second publication, aimed at applying the same architecture to estimate mammals population in images acquired from higher altitudes [9]. Despite the configuration was exactly the same of [7], the achieved results were far below, with an average precision of 0.28. The authors claimed that the application of a non-maxima suppression algorithm might have improved the whole pipeline performance by neglecting the objects overlapping for more than 75%. However, this post-processing refinement led to a negligible improvement, with a final mAP of 0.3.

III. METHOD

This paper took inspiration from [7], moving forward by exploring the possibility to perform the classification task onboard a UAV. This step allows to optimise the patrolling activity, by letting a fleet of UAV to move autonomously, to get closer to targets not clearly identified or to monitor areas in which the population density is higher. According to the project requirements, the detector should be able to distinguish among ten marine objects: sharks, whales, dolphins, surfers, swimmers, boats, buoys, rubbish, turtles and rays.

A. *Dataset creation, augmentation and filtering*

Since no relevant aerial images of marine targets were available, the best solution was to create a customized database from scratch, by manually labelling subsequent video frames. This operation was performed by modifying the code available at [10]. Essentially, the original script allowed to track an object (assigning the corresponding label tag) among similar subsequent frames. However, the original tracker (Kernelized Correlation Filter (KCF)) was replaced by Channel and Spatial Reliability Tracker - Discriminative Correlation Filter (CSRT), because it guaranteed higher precision in terms of both localization and label shape modification [11] and [12]. Moreover, other smart functionalities were added, e.g. the possibility to export and convert data in additional formats and to stop the tracking propagation. Furthermore, the target dimension was calculated by assuming a pinhole camera model and then it was compared to the one that would be seen from the operational UAV altitude (about 25-30 meters). This process was functional at removing too small targets, that were no representative of the objects seen in a real case. The final list of tagged images passed through an augmentation tool, developed by using a function written with the OPENCV library, that applied a pipeline of random geometrical (e.g. vertical/horizontal flip, image rotation and shear) and radiometric (Blur, Noise, modification of image contrast, hue, saturation and brightness) transformations to the original image. Table I shows the distribution of the 115.776 images among the classes. The total number of tags is 183.196 (some images have more than one object) and 26.548 pictures don't show any tag (only background). The dataset was finally split in three parts: 70% for training (to compute the weights), 10% for validation (to test the weights and define the update strategy), 20% for testing (to verify the final detector performance).

B. *Detector training*

Once the dataset has been created, it was used to train the CNN to learn how to locate and classify the marine objects. This section analyses at first the main features of the Faster R-CNN, Single Shot Detector (SSD) and You Only Look Once (YOLO), that were trained in Tensorflow [13] and Darknet

TABLE I
CLASSES DISTRIBUTION

CLASS	ELEMENTS
Shark	23.527
Whale	10.156
Dolphin	48.690
Turtle	16.574
Ray	10.243
Swimmer	9.021
Surfer	39.727
Boat	9.236
Rubbish	307
Buoy	15.715
Background	26.548

environments [14], [15]. Faster R-CNN guarantees better results in terms of classification precision and object location accuracy by the use of a two steps algorithm, that first localise relevant regions of interest (by the use of a Region Proposal Network (RPN) network that is nested between convolutional and pooling layers) and then classifies the objects within them [16]. Nevertheless, it pays the precision in terms of processing time, despite the huge improvements achieved from the original version, in which the regions of interest were identified by a separated Selective Search algorithm [17]. On the contrary, SSD and YOLO work as single step algorithms, in which the same neural network performs both localization and classification at the same time, resulting typically one order of magnitude faster than the former method, with a slight precision reduction. In particular, SSD optimises the detection for different object size, starting from the consideration that deeper networks analyse features coming from larger areas, whereas shallow ones are representative of smaller targets. So that, SSD performs a multi-layer classification [18] by assuming pre-defined tag shape (the anchors) that have larger size as the network get deeper. Similarly, YOLO is another project that shows a great tread-off between accuracy and speed. As faster R-CNN, it was developed over the years, dealing with different user requirements and including several innovations, e.g. the predefined anchor shape, the usage of 1x1 filters (taking inspiration from [19]) and the multi-layer classification that were used in [18]. However, YOLO is innovative because it takes the entire image (it is the input for the CNN) ,but it analyses specific portions that are defined *a-priori* with a grid, saving a considerable amount of time compared to selective search algorithms. In the end, different YOLO configurations exist. As rule of thumb, the higher the number of convolutional layers, the better the classification and localization precision, but the slower the algorithm. YOLO has three main versions, with 13, 19 and 53 convolutional layers respectively [20], [21], [22]. Modifications to these core configurations were developed to deal with specific needs, e.g. deeper networks that include the Feature Pyramid Pooling technique to classify very small objects [23], tiny networks to save computational time [15], Word-Tree approach to handle hundreds or thousands of classes [21]. In order to compare different architectures and configurations, the 'Transfer Learning' technique from pre-trained networks allowed to enhance the performance at the beginning of the training, providing steeper learning slopes and converging sooner and with higher asymptotes [24]. This methodology allowed to train YOLOv3, Tiny-YOLOv3, SSD and Faster R-CNN on the same dataset and to compare the respective performance.

C. Detector-tracking integration

In order to save computational time, a strategy to combine both detection and tracking was studied. This topic was initially analysed in [25] and [26], with encouraging results. The former proposed a detection process carried out after a fixed number of frames. In between, a tracking algorithm is responsible to find the new position of the previously identified object. Additional detections are triggered by the measurement of the resemblance of the tracked object in two subsequent frames. The latter compared the bounding box of two subsequent frames along with a piece of the surrounding area. By the use of a Matching Filter, if the principal component changes over the images, a new detection is carried out to update the saliency map. However, this method propagates the object position by using an adapted version of the particle filter, so that the effective speed improvement is not remarkable. Moreover, any of these techniques takes into account additional targets, that would not be identified neither by the tracking algorithm or by the trigger method. Indeed, UAV scenarios are highly challenging, because of multiple targets and objects that could suddenly move in or out of the camera field of view. For these reasons, the proposed detection-tracking algorithm shall be flexible to different scenarios, by measuring:

- A global frame similarity, to take into account significant changes between two subsequent frames;
- A local frame similarity, to evaluate variations within small kernels. To deal with the variations induced by camera motion, scene motion and in general not relevant motions, the algorithm considers the maximum value of the local similarity in the frame;
- The bounding box confidence, assessed by the Intersection over Union (IoU) of two subsequent frames, with the assumption that both the object motion and its shape modification have a dynamic that is slower than the camera frame-rate.

The frames similarity is inspired to the change detection concept and, specifically, it is implemented by the use of the Structural SIMilarity (SSIM) described in [27]. This index compares the variation of the luminance (l), the contrast (c) and the structure (s) of the reference and the candidate frame. The following equations describe how the SSIM is computed.

$$l(x,y) = \frac{2(1+R)}{2(1+R)^2 + \frac{C_1}{\mu_x^2}} = \frac{2\mu_y^2}{\mu_x^2 + \mu_y^2 C_1} \tag{1}$$

$$c(x,y) = \frac{2\sigma_x \sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \tag{3}$$

in which R in Eq. 1 is referred to the luminance change with respect to the background luminance, the parameters C_1,C_2 and C_3 are tuning constants, σ_x and σ_y are representative of the standard deviation, σ_{xy} of the covariance, μ_x and μ_y are the mean values of the reference and new images respectively. By multiplying the equations together, the SSIM is therefore defined:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C1)(2\sigma_{xy} + C2)}{(\mu_x^2 + \mu_y^2 + C1)(\sigma_x^2 + \sigma_y^2 + C2)} \tag{4}$$

The output matrix has hence the same dimension of the input image and assumes values equal to one if there is a total similarity, otherwise values smaller than one or even negative. The global index is computed by taking the mean value of this matrix. The mean has been preferred to the median because the latter is less sensitive to outliers. Regarding the local index, a Max Pooling (32x32-s32) is aimed at neglecting irrelevant pixel oscillations and returning the most similar point in the kernel. Figure 1 shows the flowchart of the algorithm that was implemented in order to be independent from the specific detector (coming from the CNN training) or tracker (implemented with the OPENCV library). Basically, after the first frame in which the detector is applied, for the following frames the OPENCV tracker follows the objects originally identified. At this point, the pipeline takes advantage of a fixed step exit strategy (i.e. if no relevant changes in the scenario are detected, after a fixed number of frames the pipeline performs a new detection), but for each couple of adjacent frames, it checks if the targets track is moving too fast or the target shape is changing during the time (by measuring the intersection of the bounding boxes) and the local and global image similarity. The identification of the thresholds that trigger a new detection, along with the comparison of the OPENCV algorithms is shown in Section IV.

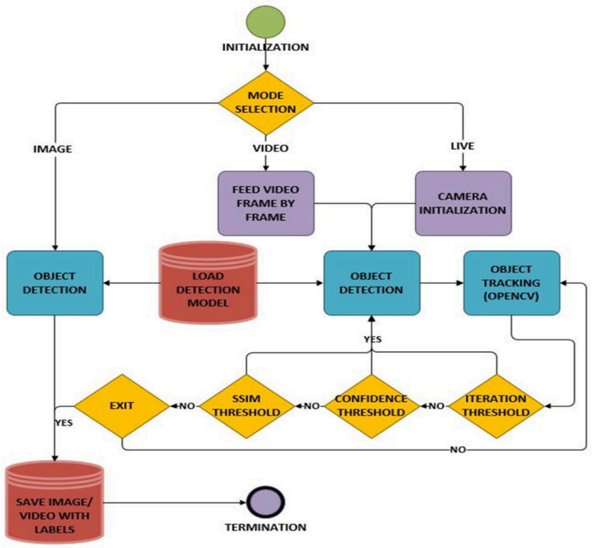

Fig. 1. Adaptive Detection-Tracker Flowchart

D. Test-bed integration

The algorithm has been tested on a UAV that was assembled at Cranfield University with off the shelf equipments. The flight control was assured by a radio controlled Pixhawk board (for this specific test no autonomous activities were foreseen). The video processing was independently performed by a dedicated GPU, a Jetson NANO, connected to a Raspberry Pi Camera v2 that provided 1920x720 frames at about 20 FPS. The whole system was powered by a 6 cells Li-Po battery by means of a 5V power regulator. The GPU was set up to process the video steaming in real time, saving relevant data in a text file for analysis purposes. Since around Cranfield University no marine environment was available, a specular detector was trained to identify pedestrians, bikes, cars and other similar ground targets. Indeed, the aim of this test was only to evaluate the processing time and therefore the operational feasibility of this system in a realistic scenario.

IV. Results

The detector evaluation and comparison was carried out considering the classification precision (mAP), the localization accuracy (IoU) and the algorithm speed (FPS). All algorithms were tested on the Deep Learning Training Server Lenovo Thinkstation P920, equipped with an Intel Xeon(R) Silver 4108 CPU (32 cores at 1.80 GHz), 64 GB of RAM and 2 GPU NVidia GeForce GTX 1080 Ti with 11.264 MB of available memory for graphical and tensor operations. Each neural network re-training (with 'Transfer Learning') took about 1-3 days, depending on the its architecture, the amount of memory usage, the amount of parallel processes ongoing at the same time.

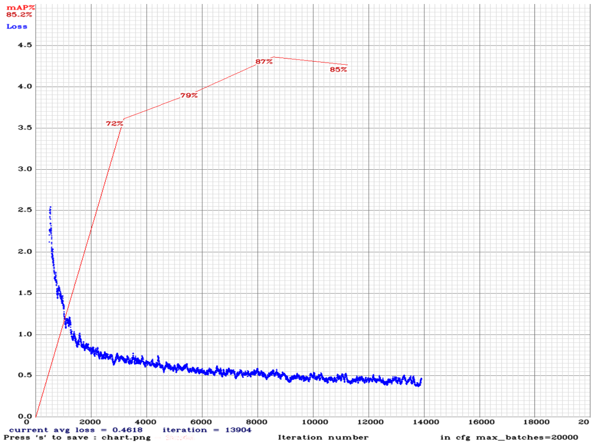

Fig. 2. Learning Curve and mAP Computation - Tiny-YOLOv3

Figure 2 shows the training output of one of the investigated YOLO configurations, specifically the blue curve represents the learning curves, i.e. the error computed once the model is applied on the training dataset, whereas the red discontinuous curve represents the mAP computed on the validation dataset.

TABLE II
TRACKER COMPARISON

	MDNFLOW	KCF	MOSSE	MIL	BOOSTING	CSRT
FPS	6.86	7.21	7.03	4.31	5.24	5.44
IoU	0.48	0.55	0.52	0.53	0.58	0.59

It is worth to point out that for each detector architecture, several model configurations were tested by fine-tuning the hyper-parameters, e.g. the number and the size of the anchors, batch and subdivision numbers, the learning rate, the input image resolution, the momentum... Finally, the best candidates for each architecture were compared on the Test dataset. Specifically, Figure 3 shows the results achieved for IoU greater than 0.5. Faster R-CNN and YOLO were similar in terms of mAP (about 0.90), whereas SSD returned lower precisions. However, even if the boxes localization was similar, from a qualitative perspective Faster R-CNN gave results closer to the real object shape. Finally, Tiny-YOLOv3 outperformed speed comparison, being at least 5 times faster than the any other architecture, with a negligible accuracy degradation. Moreover, the values shown in Figure 3 were obtained on a Python pipeline, but Tiny-YOLOv3 could run natively on Darknet (in C) achieving top speed of 30 FPS.

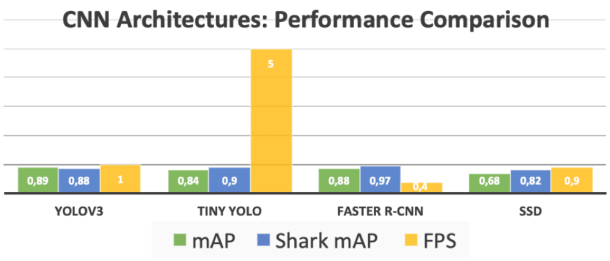

Fig. 3. CNN Architecture comparison

Regarding the integration with the tracker, the hyper-parameters described in the previous chapter were tuned in order to find the best configuration in terms of tracking accuracy and algorithm speed. The baseline results were obtained by processing an entire marine video with the Faster R-CNN detector and it was used to assess the quality of the proposed algorithm, i.e. without using any tracker, but only subsequent detections. Table III shows a comparison of the different OPENCV tracker performance. During each test, two parameters were measured: the IoU with the "Ground Truth" (in this case represented by the pure detections) and the pipeline speed. The video that was used for the test was quite challenging, since there were several changes of the scene, fast movements, multiple objects to detect and track. According to Table III, KCF provides almost the top accuracy (only CSRT was slightly more accurate), but it is by far the fastest tracker among the ones implemented in the OPENCV library, so that it is the best solution for low performance hardware systems. In Table III, the analysis is focused on the parameters that were implemented to balance the alternation between detection and tracking. However, an alternative strategy could be to use a fixed interval step, but the results with the control logic implemented show that significant improvements are indeed possible. Each block represents a test in which only one parameter changes its value (the first row of each block shows the values of the Interval Step, Local SSIM Threshold..). The assessment takes into account the pipeline speed and the overlapping with the "detector-only" case. The first two elements of the table include the performance comparison with and without the control low (new detection after a fixed amount of frames). The proposed strategy allows, with a negligible speed reduction, to significantly improve the accuracy, because the detections are performed only when they are effectively needed. It is worth observing that the control law allows to keep the interval parameter to 100 Frames Per Second (FPS), providing an higher IoU with respect to the case in which there is classic 30 frames interval logic. This can explained because in most of the cases a new detection is triggered after more than 30 frames, because the scene is quite stable, but there are unpredictable conditions in which the scene evolves quite fast and a fixed strategy is not enough to deal with it. Furthermore, it is possible to appreciate how sensitive the pipeline is with respect to the Local SSIM threshold. On the contrary, the Global SSIM seems to be useless, according to the table results, because it does not change significantly neither FPS nor IoU. However, since the video used for the evaluation had very fast scene changes and various objects coming in the scene, such a parameter was good to immediately perform a detection, without waiting for other control laws to trigger it. Also the IoU threshold plays a significant role in this process, because it measures how different is the same bounding boxes in two consecutive frames. However, this parameter might find beneficial the integration with the carrier motion data in order to better understand which movements are self generated, so that it might be possible to better isolate moving targets from the background or still objects. Hence, future implementations of the algorithm might take into account this parameter to perform more accurate balancing. According to the results in Table III, the parameters chosen by the author are: IoU=0.5, Interval Steps=50, Local SSIM Threshold=0.55, Global SSIM Threshold=0.7. By testing this pipeline with the Tiny-YOLOv3 algorithm we found a speed improvement of about 44% (7.22 FPS with the Python algorithm). It is true that similar performance was achieved in [7], but in this case the number of classes is more than doubled, Faster-R-CNN detector, that is the CNN architecture implemented in [7], was 5 times slower than Tiny-YOLOv3. Moreover, as shown before, the detector only reaches the top speed of 30 FPS in its native C implementation. On-board the test-bed identified no relevant operational limitation, but the algorithm speed decreased due to the limited computational power of the selected computational board, the Jetson NANO. In terms of numbers, the Python pipeline achieved about 0.5 FPS instead of 7. However, it is quite encouraging that the C Darknet model achieved about 3.4 FPS, so that it is essentially a problem of coding language. These numbers are still non enough for Real-Time applications, but might be used to analyse low dynamic targets or lower resolution images (since the lower

TABLE III
ADAPTIVE DETECTION TRACKER - PARAMETERS EFFECTS

No Control Logic			
Interval Step	30	50	100
FPS	5.23	5.72	5.89
IoU	0.54	0.45	0.42
Control Logic			
Interval Step	30	50	100
FPS	5.12	5.49	5.79
IoU	0.62	0.56	0.55
Local SSIM Threshold	0.8	0.7	0.6
FPS	2.69	4.27	4.97
IoU	0.68	0.62	0.59
Global SSIM Threshold	0.9	0.8	0.7
FPS	5.15	5.13	4.91
IoU	0.60	0.61	0.62
IoU Threshold	0.6	0.55	0.4
FPS	2.15	5.13	5.4
IoU	0.81	0.54	0.49

the resolution, the faster the algorithm). Nevertheless, by using more powerful processing board (e.g. the Jetson XAVIER) this limitation might be easily overcome.

V. CONCLUSIONS

This paper demonstrated how the problem of object detection might be addressed with a decentralized system, in which a UAV agent flies autonomously recording and processing video in Real-Time. Current state of the art solutions are based on human in the loop or centralized system architectures, in which the ground station represents the bottleneck, since it has to process video-streaming data coming from one or more UAV agents. On the contrary, the system presented in this paper can be integrated on a completely autonomous UAV, simplifying the data transfer management and making the system architecture easily scalable. The results achieved are highly encouraging and the following step of this research is to forward the target location to higher order intelligence in order to manage autonomous guidance and navigation. Moreover, the implementation of the detection-tracking system in C instead of python shall guarantee better performance.

ACKNOWLEDGMENT

The authors would like to thank the project sponsor, THHINK, for suggesting the topic and taking part to the initial brainstorming phase of the research.

REFERENCES

[1] J. G. West, "Changing patterns of shark attacks in Australian waters," *Marine and Freshwater Research*, vol. 62, no. 6, p. 744, 2011.

[2] F. Maire, L. Mejias, A. Hodgson, and G. Duclos, "Detection of dugongs from unmanned aerial vehicles," *IEEE International Conference on Intelligent Robots and Systems*, no. November, pp. 2750–2756, 2013.

[3] L. Mejias, G. Duclos, A. Hodgson, and F. Maire, "Automated marine mammal detection from aerial imagery," *IEEE/MTS Proceedings of 2013 OCEANS - San Diego*, pp. 1–5, Sep. 2013.

[4] K. Byles, "Automated shark detection using computer vision," 2016, university of Southern Queensland - Faculty of Health, Engineering & Sciences. Bachelor Thesis.

[5] J. Lopez, J. Schoonmaker, and S. Saggese, "Automated detection of marine animals using multispectral imaging," *IEEE/MTS Proceedings of 2014 OCEANS - St. John's*, pp. 1–6, 2015.

[6] S. Gururatsakul, D. Gibbins, D. Kearney, and I. Lee, "Shark detection using optical image data from a mobile aerial platform," *International Conference Image and Vision Computing New Zealand*, pp. 1–8, 2010.

[7] N. Sharma, P. Scully-Power, and M. Blumenstein, "Shark Detection from Aerial Imagery Using Region-Based CNN, a Study," in *AI 2018: Advances in Artificial Intelligence*, vol. 3339. Springer International Publishing, 2018, pp. 224–236.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[9] M. Saqib, S. Daud Khan, N. Sharma, P. Scully-Power, P. Butcher, A. Colefax, and M. Blumenstein, "Real-Time Drone Surveillance and Population Estimation of Marine Animals from Aerial Imagery," *International Conference Image and Vision Computing New Zealand*, vol. 2018-Novem, pp. 1–6, 2019.

[10] J. Cartucho. (2018) GITHUB Repository - OPEN LABELING. https://github.com/Cartucho/OpenLabeling, Accessed on 2019/05/20.

[11] J. A. Torregrosa Olivero, C. María Burgos Anillo, J. P. Guerrero Barrios, E. Montoya Morales, E. J. Gachancipá, and C. Andrés Zamora de la Torre, "Comparing state-of-the-art methods of detection and tracking people on security cameras video," in *2019 XXII Symposium on Image, Signal Processing and Artificial Vision (STSIVA)*, April 2019, pp. 1–5.

[12] A. Rosebrock. (2018) OpenCV Object Tracking. https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/, Accessed on 2019/07/10.

[13] Google Brain Team. (2015–2019) TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/, Accessed on 2019/06/01.

[14] J. Redmon, "Darknet: Open Source Neural Networks in C," 2013–2019, http://pjreddie.com/darknet/, Accessed on 2019/06/01.

[15] B. Alexey, "GITHUB Repository - Darknet," 2019, https://github.com/AlexeyAB/darknet, Accessed on 2019/05/20.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 91–99.

[17] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision*, 2013.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*. Springer International Publishing, 2016.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[20] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2015.

[21] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525, 2016.

[22] Redmon, J. and Farhadi, A., "Yolov3: An incremental improvement," in *arXiv*, 2018.

[23] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944, 2016.

[24] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques*. Information Science Reference - Imprint of: IGI Publishing, 2009.

[25] S. P. Bharati, S. Nandi, Y. Wu, Y. Sui, and G. Wang, "Fast and Robust Object Tracking with Adaptive Detection," in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov 2016, pp. 706–713.

[26] B. Nemade and V. A. Bharadi, "Adaptive automatic tracking, learning and detection of any real time object in the video stream," in *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, Sep. 2014, pp. 569–575.

[27] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004.