Real-Time Drone Anti-Collision Avoidance Systems: an Edge Artificial Intelligence application

Iyad LAHSEN-CHERIF[1], Huan LIU[1], Catherine LAMY-BERGOT[1]

[1] Thales Communications & Security
4 avenue des Louvresses, 92230 Gennevilliers, France.
{name.surname}@thalesgroup.com

ABSTRACT

Collision Avoidance (CA) is one of the most challenging and fundamental problems in the Unmanned Aerial Vehicle (UAV) ecosystem. The unmanned version of Airborne Collisions Avoidance Systems (ACAS-Xu) detects collision threats and informs pilots of avoidance maneuvers to perform. It makes decisions based on an optimized cost tables. However, these tables are too large and heavy ($\sim$5GB) to be integrated on the current certified avionic hardware. In this paper, we present a machine learning (ML) based approach to reduce the memory footprint of the cost tables by approximating them using a deep neural network (DNN). Training results show that the proposed approach reduces the memory footprint drastically by a factor of 500 from 5GB to 10MB while maintaining the same level of safety. Additionally, we embedded the DNN based approach on an edge artificial intelligence (AI) device and demonstrated its ability to operate on limited hardware with low processing time.

I. INTRODUCTION

In 1978, a collision between a light aircraft and a commercial airliner over San Diego, California, led the Federal Aviation Administration (FAA) to launch the Traffic alert and Collision Avoidance System (TCAS) [13]. TCAS helps pilots to avoid nearby traffic using traffic alerts (TAs) and to prevent near mid-air collisions (NMAC). In 1986, a collision over Cerritos, California, resulted in a mandate requiring certain classes of aircraft to be equipped with TCAS II, an improved version of the Collision Avoidance (CA) system providing resolution advisories (RAs) to mitigate the risk of collision. Although these systems performed well in tests, it was concluded that these systems generated a high rate of unnecessary alerts which disturb pilots and lead them to make unnecessary manoeuvres..

The FAA has initiated research and development of a new approach to Airborne Collision Avoidance System (ACAS-X) since 2008 with the main objective of detecting collision threats and informing the pilot of the vertical and/or horizontal avoidance maneuvers to perform. ACAS-X delivers several advantages over the classical TCAS systems by guaranteeing improved safety while reducing the unnecessary alert rate. The intent is that in the next few years, ACAS-X will gradually replace TCAS II.

ACAS-X system is capable of providing collision avoidance (CA) maneuvers and resolution advisories (RA) that take into account particularities of each platform (airplane, drone, helicopter) through its different variants: ACAS-Xa (active), ACAS-Xp (passive), ACAS-Xo (operations) and ACAS-Xu designed for UAV or remotely piloted aircraft systems. While this work concerns the ACAS-Xu version, it can be easily extended to other ACAS-X versions.

ACAS-Xu takes advantage from recent advances in computational techniques. It formulates the CA problem as a Markov Decision Process (MDP) and uses Dynamic Programming to generate optimized advisories. The decision logic is represented as a large numeric table known as cost table storing scores for all possible manoeuvres and it used to determine resolution advisories given to pilots. However, the cost table size grows dramatically ($\sim$5GB) with the number of state variables and makes it challenging to embed in a certified avionic hardware.

Deep neural networks (DNNs) have demonstrated state-of-the-art performance on many AI applications such as speech recognition, image classification and video prediction. Replacing the classical cost table by a DNN allows to gain in memory footprint size. Moreover, instead of indicating the avoidance maneuver by consulting a large amount of data present in the cost table, the system computes the avoidance costs using the DNN.

The contribution of this study is twofold. First, we focus on cost table compression and base our work on DNNs to approximate the classical cost table. Then, we explore DNN embedding on a dedicated edge AI device and evaluate the integration results in terms of processing time and hardware capabilities.

The rest of the paper is structured as follows: In the next section we briefly review related work. Section III describes the cost table compression and the training process. Then, in Section IV we investigate DNN embedding and evaluate the real-time inference phase. Finally, we conclude the paper and present future work in Section V.

II. RELATED WORK

A. Collision avoidance related work

Several studies have been conducted on CA systems. Most of these studies used dynamic programming and optimization tools to derive the CA decision logic.

The authors of [7] proposed an Origami based table compression approach. The main idea is to rearrange the cost table into a collection of small tables by exploiting redundancies and symmetries in the system. This approach results in acceptable safety performance. However, the compressed table representation still occupies a large memory footprint. The CA problem is formulated as a stochastic problem in the form of a multi-agent Markov decision process (MMDP) in [10] with a focus on advisories in the horizontal plane. Compared to closest-threat heuristics and an uncoordinated algorithm for collision avoidance, authors show that this method scales and resolves conflicts efficiently. New collision avoidance approaches for ACAS sXu, a variant that provides CA for small UAV (sUAV), are introduced in [3]. Using Monte Carlo simulations of three different scenarios: *(i)* manned aircraft avoiding sUAV, *(ii)* sUAV avoiding manned aircraft and *(iii)* manned aircraft and sUAV avoiding each other's, authors showed that the proposed CA scheme leads to a good safety level and optimal alerting rates compared to classical CA schemes. A DNN is used as an alternative to reduce the cost table size in [6]. Practically, the regression DNN gathers the Kinematics between the ownship and the intruder to output the associated cost vectors.

Only few literature works studied cost table reduction size using a DNN based approach and none of these works detailed the dataset generation and training processes.

B. Edge AI related work

The usage of DNN based models for real-time applications received considerable attention from both industry and academia in recent few years. This subsection reviews previous work on embedding DNNs on edge AI devices.

The processing resources usage for object detection applications is evaluated in [12]. The study is performed using a NVIDIA Jetson Nano and the You Only Look Once (YOLO) algorithm is used to identify and count objects in real-time. The same edge AI device is used to embed a pedestrian and vehicles recognition framework in [1]. The authors used convolutional DNNs for object detection tasks. The results are good pointing out that the processing time is relatively high in complex environmental conditions. The authors of [9] propose an implementation of a tiny version of the YOLO model for a real-time object detection application. The study highlights the efficiency of DNNs deployment on cheap and low power on embedded hardware such as Nvidia Jetson Nano. Several DNNs and edge computing devices for autonomous safety and security of mobile and aerial robotics are highlighted in [4]. The authors overview optimization techniques to speed up DNNs inference such as network pruning, knowledge distillation, model partitioning and data quantization.

In summary, DNNs can be embedded in hardware such as the NVIDIA Jetson Nano while maintaining good performance. However, to the best of our knowledge, no previous work studied the integration of a time sensitive application such as CA for UAVs and analyzed its performance on an AI dedicated hardware.

Variables	Min	Max	Unit	Type
Range (ρ)	11	9.6e+4	nm	float
Theta (θ)	-π	π	radian	float
Phi (φ)	-π	π	radian	float
Ownship speed (v_{own})	55	270	nmph	float
Intruder speed (v_{intr})	12	320	nmph	float
Previous action (a_{prev})	1	5	-	int
Current Action ($a_{current}$)	1	5	-	int

TABLE I: Model variables.

III. Cost Table Compression

In this section we introduce the system model, the dataset, the DNN and the training results.

A. Problem Formulation

We consider a system composed of two UAVs: the ownship, equipped with ACAS-Xu system, and the intruder, not equipped with any CA system, facing each other. The system model is shown in Figure 1, where:

- ρ (ft): the horizontal range,
- θ (rad): the relative intruder track angle,
- φ (rad): relative bearing to the intruder,
- v_{own} (ft/s): the ownship speed,
- v_{intr} (ft/s): intruder speed.

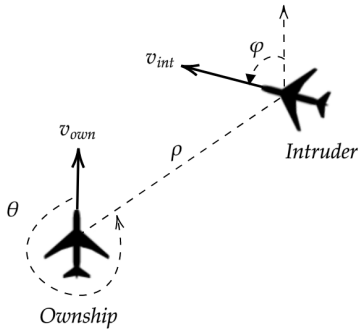


Figure 1: Ownship and intruder on the horizontal plane.

The resolution advisories and possible actions to be taken by the ownship are categorized into two types: (i) the ones with no necessary maneuvering and (ii) the ones constraining the ownship to a given turn rate. The former mainly consists of the Clear Of Conflict maneuver (COC) allowing the ownship to turn freely, while the later is composed of four advisories described as follows:

- Weak Left (WL): [1.0°/s, 2.0°/s]
- Strong Left (SL): [2.0°/s, 4.0°/s]
- Weak Right (WR): [-2.0°/s, -1.0°/s]
- Strong Right (SR): [–4.0°/s, -2.0°/s]

B. Dataset

The dataset is generated based on the classical cost table and by simulating a set of encounter models. The considered encounter models [8] are based on realistic flight trajectories simulating collision scenarios between the ownship and the intruder. The dataset generation process is shown in Figure 2 and is described as follows:

[1] Defining and running representative encounter models corresponding to a set of ownship-intruder pairs.
[2] Collecting sensors information and ownship-intruder kinematics (range, speeds, angles) to form an input vector.
[3] Calculating the cost vector based on the *classical cost table* and the input vector.

This operation is repeated periodically for each encounter model simulation. An encounter model simulation lasts 120 sec and data (input and cost vectors) is recorded every one millisecond (1ms). The dataset[1] contains 7.8 millions instances ($\sim$ 5GB of data) and it is formed by concatenating the input vector and the cost vector.

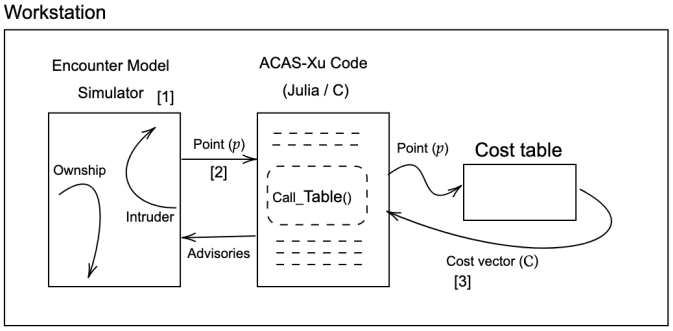


Figure 2: Dataset generation Pipeline.

The ACAS-Xu code processes the input vector and contains a function calling the cost table (call_Table()). Furthermore, this code maps the output cost vector, composed of 10 costs, to resolution advisories (COC, WL, SL...) provided to pilots.

C. Deep Neural Network model Training

We remind the reader that the objective of this section is to demonstrate that the DNN approximation of the classical cost table reduces the memory footprint without affecting the safety performance.

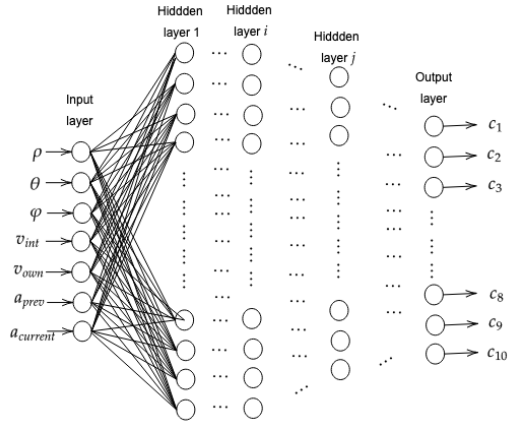


Figure 3: The proposed DNN architecture.

The proposed DNN is a Multi-Layer Perceptron (MLP) composed of an input layer of 7 nodes, 9 hidden layers and an output layer of 10 nodes. A MLP is fully connected DNN where each node is connected with all nodes of the next layer and each connection has a particular weight (w) characterizing the impact of this node on the nodes of the next layer. The objective of the DNN is to approximate the cost vector $\mathcal{C} = [c_1, c_2, ..., c_{10}]$, given an input vector $p = [\rho, \theta, \varphi, v_{own}, v_{intr}, a_{prev}, a_{current}]$, with a_{prev} is the previous resolution advisory and $a_{current}$ is the current resolution advisory. This problem is known as a *multi-regression* problem:

$$\mathcal{C} = f(p), \tag{1}$$

where f is the function to be learned.

During the training phase, the model learns network weights (w) from the training set (80% of dataset). The weights are iteratively updated using the back-propagation algorithm [5] in order to reduce the network loss and regression errors. In the inference phase, the network weights are fixed and the model is evaluated on test data (20% of dataset).

The DNN is implemented in Python using the Keras library running in the top of Tensorflow and is trained using a workstation equipped with a NVIDIA dual Titan RTX GPUs.

D. Simulation results

In this subsection, we discuss both DNN performance and operational results.

We have performed a grid search to find the optimal DNN hyperparameters values. Grid search is a tuning technique based on exhaustive search to compute the optimum values of hyperparameters. Due to lack of space, grid search details are omitted. The optimal network architecture is a DNN with 11 layers of 7, 128, 128, 128, 64, 64, 64, 32, 32, 32, 10 nodes. The optimal hyperparameters found through the grid search process are a RELU activation function for hidden layers and a sigmoid for the output layer with a learning rate of 0,1, an Adam optimizer, a normal kernel initialization and a batch size of 32.

To evaluate the DNN performance, we consider the same pipeline used to generate the dataset. The only difference is that instead of computing resolution advisories based on the classical cost table we use the trained DNN.

1) ML performance: We evaluated the DNN performance and its ability to generalize using the R^2 (R-squared) metric defined as:

$$R^2 = 1 - \frac{\sum_i (\mathcal{C}_i - f(p_i))}{\sum_i (\mathcal{C}_i - \hat{C})} \tag{2}$$

where p_i denotes the input vector of instance i, $f(p_i)$ the cost vector predicted by the DNN for instance i, $\mathcal{C}_i$ the ground truth cost vector of instance i and $\hat{\mathcal{C}}$ the average of all predicted cost vectors. The R^2 metric measures how well the DNN model fits the classical cost table data. Usually, it ranges between 0 and 1 and the closer it is to 1, the more accurate the regression of the model is.

Figure 4 shows the R^2 metric convergence. The R^2 converges quickly and reaches a score of 99% in less than 50 epochs. The same convergence speed is observed for the MSE metric which converge to 0 after few epochs of training.

[1]The dataset is available on request from the authors.

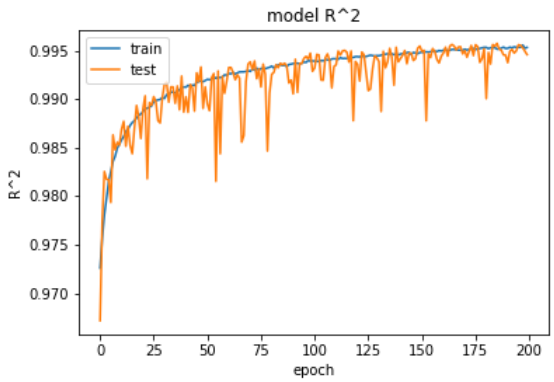


Figure 4: R^2 vs Epochs.

2) Safety performance: To derive safety performances, we conducted extensive simulations of 5000 encounter model scenarios and computed the anti-collision advisories using both DNN and classical cost table.

The Near Mid Air Collision (NMAC) rate is used to evaluate the safety of the DNN equipped system. It occurs when two aircrafts come within 500 feet (∼150m) horizontally. Simulation results show that the NMAC rate of DNN is lower, and hence better, than the NMAC rate of the classical cost table by 0.13% (0.0013). Therefore, it is guaranteed that resolution advisories given by the DNN do not degrade the safety of the system.

Furthermore, the recommended avoidance manoeuvres are 99.7% identical to the manoeuvres proposed by the classical cost table, ensuring a strong coherence between the two approaches.

It can be concluded that approximating the classical cost table by a DNN reduces drastically the size by a factor of 500 without affecting safety performance.

In the next section, we study the feasibility of embedding the trained DNN in an edge AI device.

IV. Anti-Collision Model Embedding: Edge AI Application

A. Materials Description: NVIDIA Jetson Nano

We used a NVIDIA Jetson Nano[2] to embed the DNN and to accelerate the inference process. The NVIDIA Jetson Nano is a low power Graphical Processing Unit (GPU) offering high processing capabilities. It is a compact size device (69,6mm x 45mm), equipped with a quad-core ARM Cortex-A57 CPU, a 128 CUDA core Nvidia Maxwell architecture GPU, a 2GB of RAM and can work in two power modes at 5 Watts and 10 Watts. The NVIDIA Jetson Nano, shown in Figure 5, is the lightest model of Jetson series, with 140 grams and peak performance of 472 GFLOS[3]. An advantage of the NVIDIA Jetson Nano is it compatibility with the most popular machine learning frameworks such as TensorFlow and PyTorch and hence making deployment of AI based models an easy task.

[2]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

[3]Giga Float OPerations per second

Figure 5: NVIDIA Jetson Nano device.

B. Scenario pipeline

The same pipeline used in the previous section for dataset generation and performance evaluation is used in this section too. However, instead running experiments on the workstation, the pipeline is embedded on the NVIDIA Jetson Nano as shown in Figure 6.

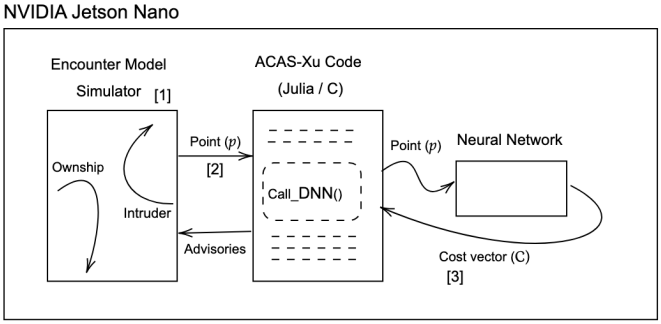
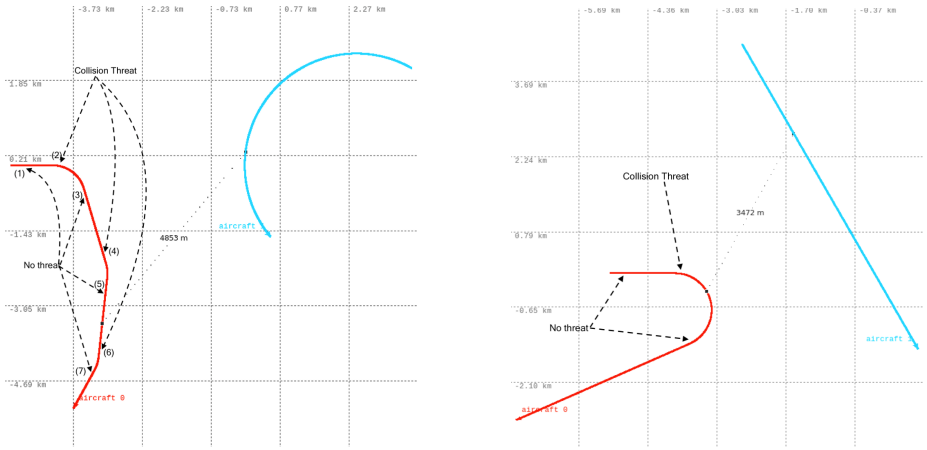


Figure 6: Collision Avoidance Pipeline.

We consider two different scenarios where the ownship is equipped with the ACAS-Xu capabilities on the horizontal plane and tries to avoid the intruder. In scenario 1, the intruder has a curved path while in scenario 2 the intruder has a direct path as illustrated in Figure 7. Both scenarios last 120 seconds.

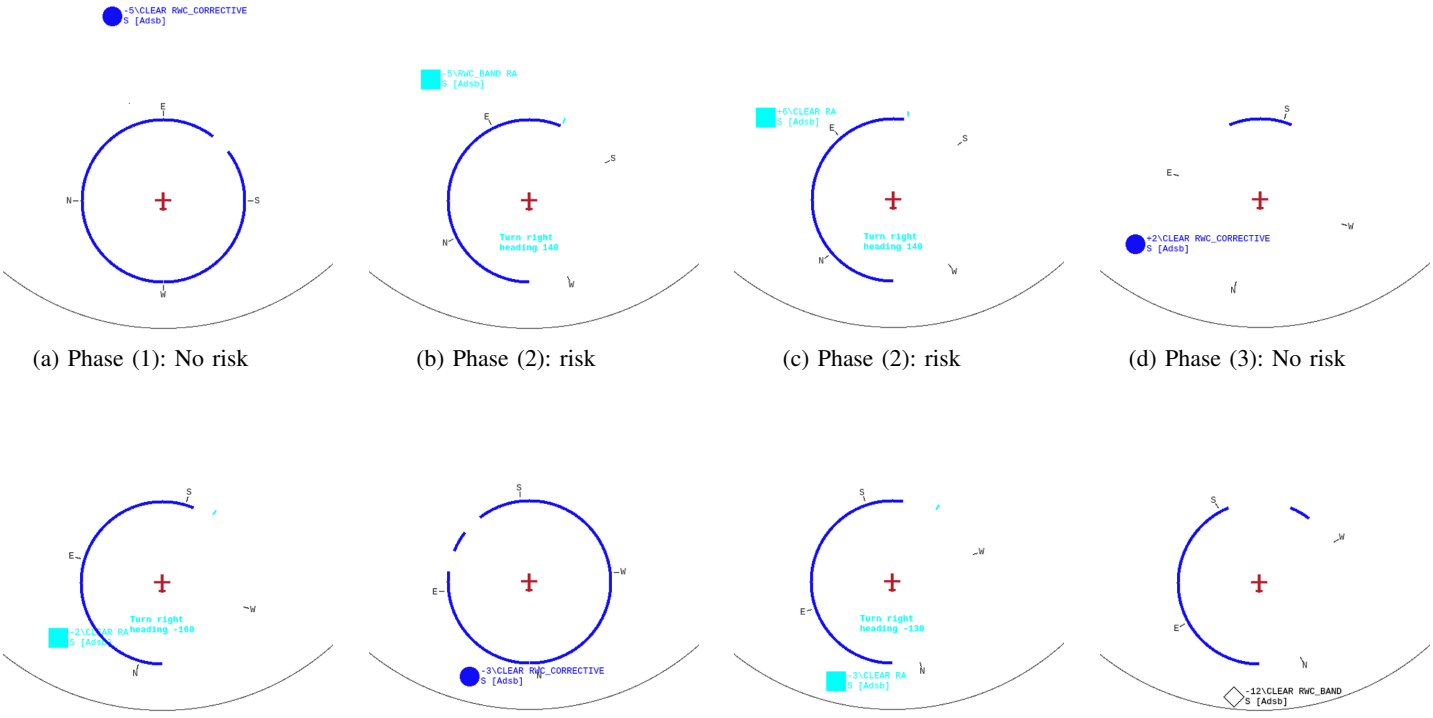
As shown in Figure 7a, scenario 1 can be decomposed into 7 phases as follows: In phase *(1)* no threat is raised. A threat is raised in phase *(2)* when the intruder starts approaching the ownship (Fig. 8b and Fig 8c). The ACAS-Xu proposes a resolution advisory to the ownship to turn right. During phase *(3)* the threat is withdrawn and no maneuver is proposed to the ownship (Fig. 8d). Due to the curved path of the intruder, it enters a risk area and hence raises a new collision threat. The ACAS-Xu advises the ownship to turn right during phase *(4)* (Fig. 8e). In phase *(5)* the threat is withdrawn (Fig. 8f). During phase *(6)*, the intruder gets closer again to the ownship and new advisories are released by the ACAS-Xu (Fig. 8g). Finally, the risk of collision is withdrawn and simulation ends during phase *(7)* (Fig. 8h). The state of the intruder and the advisories given by the ACAS-Xu system are detailed in Figure 8.

Furthermore, the minimum distance between the ownship and the intruder is 4853m respecting the NMAC separation distance.



(a) Scenario 1: Curved intruder path. (b) Scenario 2; Direct intruder path.

Figure 7: Scenarios illustrations: Aircraft 0 (in red) is the ownship and Aircraft 1 (in blue) is the intruder.



(a) Phase (1): No risk (b) Phase (2): risk (c) Phase (2): risk (d) Phase (3): No risk

(e) Phase (4): risk (f) Phase (5): No risk (g) Phase (6): risk (h) Phase (7): No risk

Figure 8: Scenario 1 phases: ownship in red and intruder in blue.

C. Embedding results

Having a low processing time is an important advantage when designing DNNs based applications in embedded systems. In this subsection we evaluate the simulation (inference) time and processing capabilities of the considered edge AI device.

We implemented 5 different versions of the ACAS-Xu DNN using various languages: Julia [2], CUDA Julia using libraries such as Flux.jl, C, CUDA C and an optimized CUDA C version. CUDA [11] is a parallel computing platform and programming model for general computing on graphical processing units (GPUs). It reduces the time to perform compute-intensive tasks, by allowing workloads to run on GPUs. It can be seen from Table II that the processing time is reduced using C language compared to Julia. Additionally, CUDA speeds up the DNN calculations for both C language and Julia versions. In fact the processing time of the played scenario is 360 sec using C language, and it is reduced to 132 sec using a CUDA C code for scenario 1 and it is 1303 sec for Julia reduced to 794 sec using CUDA Julia for the same scenario.

The optimized C CUDA code version is a DNN performing operation in half-precision format. This gives a significant acceleration over normal C CUDA and normal C codes. Converting DNN weights and bias from single precision floats

(FP32) to half precision floats (FP16) allows to reduce the number of bytes accessed, thus reducing the execution time. Naturally, the execution time of scenario 1 is higher than scenario 2 because the collision threat is raised 3 times and hence the DNN is solicited 3 times (each collision threat until the threat is withdrawn).

DNN implementation	Scenario 1	Scenario 2
Julia	1303 sec	1200 sec
Julia CUDA	794 sec	700 sec
C	360 sec	300 sec
C CUDA	132 sec	133 sec
Optimized C CUDA	55 sec	50 sec

TABLE II: Simulation times.

Furthermore, we measured the NVIDIA Jetson Nano performance in terms of GPU, CPU, RAM usages and equipment temperature using the Jetson Stats Tool[4]. As reported in Table III, the average GPU usage over time rises from 4% to 30 % for scenario 2 and 40% for scenario 1. In fact, the collision threat is raised 3 times in scenario 1 vs 1 time for scenario 2 and hence the DNN solicits GPU capabilities for a longer period when scenario 1 is played. The RAM memory and CPU usage changed from 12% to 90% and from 5% to 96% respectively, and remained at the same level during the whole simulations. Device temperature is highly correlated to GPU usage and it is 41°C for scenario 1 and 39°C for scenario 2.

Scenario	Nothing	Scenario 1	Scenario 2
GPU (%)	4%	40%	30%
CPU (%)	5%	96%	96 %
RAM (%)	12%	90%	90%
Temperature (°C)	37.5	41	39

TABLE III: NVIDIA Jetson Nano performance.

V. Conclusion and Future work

Airborne Collision Avoidance Systems for unmanned platforms (ACAS-Xu) is a promising solution for protection against collision threats. In this work we proposed a DNN based approach to approximate the classical cost table used to provide resolution advisories. Furthermore, we embedded the trained DNN on a NVIDIA Jetson Nano and showed through simulations that the processing time can be significantly accelerated using CUDA optimization.

Certifying edge AI devices to be integrated in flying avionics is an important and challenging task. Our future works lie on explanation and verification advisories provided by the DNN.

Bibliography

[1] Luis Barba-Guaman, José Eugenio Naranjo, and Anthony Ortiz. "Deep Learning Framework for Vehicle and Pedestrian Detection in Rural Roads on an Embedded GPU". In: *Electronics* 9 (2020).

[2] Jeff Bezanson et al. "Julia: A fast dynamic language for technical computing". In: *arXiv preprint arXiv:1209.5145* (2012).

[3] John Deaton and Michael P Owen. "Evaluating Collision Avoidance for Small UAS using ACAS X". In: *AIAA Scitech 2020 Forum*. 2020, p. 0488.

[4] Efstathios Faniadis and Angelos Amanatiadis. "Deep Learning Inference at the Edge for Mobile and Aerial Robotics". In: *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2020, pp. 334–340. DOI: 10.1109/SSRR50563.2020.9292575.

[5] Robert Hecht-Nielsen. "Theory of the backpropagation neural network". In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[6] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. "Deep Neural Network Compression for Aircraft Collision Avoidance Systems". In: *CoRR* abs/1810.04240 (2018). eprint: 1810.04240.

[7] Kyle D Julian et al. "Policy compression for aircraft collision avoidance systems". In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE. 2016, pp. 1–10.

[8] MJ Kochenderfer et al. "Correlated encounter model for cooperative aircraft in the national airspace system version 1.0". In: *Project Report ATC-344, Lincoln Laboratory* (2008).

[9] Vittorio Mazzia et al. "Real-Time Apple Detection System Using Embedded Systems With Hardware Accelerators: An Edge AI Application". In: *IEEE Access* 8 (2020), pp. 9102–9114.

[10] Hao Yi Ong and Mykel J Kochenderfer. "Markov decision process-based distributed conflict resolution for drone air traffic management". In: *Journal of Guidance, Control, and Dynamics* 40.1 (2017), pp. 69–80.

[11] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[12] Sebastián Valladares et al. "Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application". In: *Intelligent Human Systems Integration 2021*. Ed. by Dario Russo et al. Cham: Springer International Publishing, 2021, pp. 343–349. ISBN: 978-3-030-68017-6.

[13] T. Williamson and N.A. Spencer. "Development and operation of the Traffic Alert and Collision Avoidance System (TCAS)". In: *Proceedings of the IEEE* 77.11 (1989), pp. 1735–1744. DOI: 10.1109/5.47735.

[4] https://pypi.org/project/jetson-stats/