Modular Neural Network Navigation for Autonomous Nano Drone Racing

Federico Magri, Robin Ferede, Stavrow Bahnam, Christophe De Wagter, Guido C.H.E De Croon

***Abstract*—In this study, we present a first step towards a cutting-edge software framework that will enable autonomous racing capabilities for nano drones. Through the integration of neural networks tailored for real-time operation on resource-constrained devices. A lightweight Convolutional Neural Network, with the Gatenet architecture, is adjusted for reduced computational demand and is successfully deployed on a GAP8 processor at a rate of 16Hz. This network provides gates' size and location data for the subsequent positioning algorithm. A second neural network, trained through reinforcement learning, governs the drone's guidance and control systems, demonstrating a remarkable rate of 167Hz on an STM32F405 processor. The attitude rates and thrust outputted by this network are then fed to an attitude rate PID controller.**

The research shows that state-of-the-art neural networks for drone racing can be deployed on nano drones, despite their limited processing power. Nonetheless, the study demonstrated specific limitations, such as the perception network's sensitivity to white pixels in the image reducing its effectiveness when light sources are present in the scene. These findings underscore the importance of dataset composition and the need for diverse training scenarios to enhance the neural network's generalizability and performance in real-world applications.

***Index Terms*—Reinforcement learning, Convolutional neural network, Quantization, Drone Racing**

I. Introduction

DEEP Learning has seen incredible progress over the last years, with notable advancements propelled by the integration of reinforcement learning and deep neural network architectures. These methods have proven their worth by surpassing human experts in several competitive areas, including games like Go [1], Starcraft [2], and chess [1]. Despite these successes, the application of Artificial Intelligence(AI) has remained predominantly within the bounds of simulations and controlled gaming contexts, lacking the unpredictable variables present in real-world settings.

The world of drone racing, where trained pilots maneuver quadcopters at speeds exceeding 100 km/h, has recently been recognized as a prime environment for testing and enhancing AI systems. The Alphapilot autonomous racing competition in 2019 was a showcase of the cutting-edge research being conducted in this domain. However, human pilots were still significantly faster, with drones operated by AI lagging by a factor of at least two [3]. A paradigm shift occurred with the introduction of the Swift system, which, powered by a Nvidia Jetson TX2, successfully outperformed multiple professional human pilots for the first time in history [4].

With the rise of edge artificial intelligence, complex algorithms can be run on smaller drones [5]. Nano drones, with diameters under $10cm$, are an agile and versatile robotic platform employed in cases where size is of the essence. Spanning from aerial inspection tasks in narrow places [6] to human-robot interaction tasks [7]. Given the limited flight time from the battery size, the faster these drones can fly the more efficient the application is.

This article describes a system that can be used to autonomously race on nano drones, using artificial intelligence. Two lightweight deep neural networks are introduced in this article. The perception neural network detects the gates and is trained with supervised learning. The output of this network is the center and size of the gate similar to an approach taken by Pham et al. [8]. These outputs are then used to estimate the relative position. The second neural network is for guidance and control, it commands the attitude rates and thrust of the drone. This network is trained via reinforcement learning. Both networks are accelerated via tuning of the number of parameters and quantization.

The three major contributions of this article are the successful development of a neural network capable of accurately locating racing gates at very low image resolutions. The deployment of this network on the AI-Deck, achieving inference speeds sufficient for a nano drone to fly. Finally, the implementation of a lightweight guidance and control neural network, trained via reinforcement learning, on the flight controller of a Crazyflie drone.

II. Related work

Advanced algorithms such as Visual Inertial Odometry [9] or feature-based Simultaneous Localization And Mapping (SLAM) [10] are unviable due to their intensive computational demands. Similarly, the application of control strategies, like Model Predictive Control [11], are discarded because of their computational intensity.

Consequently, we move towards employing Neural Networks, prioritizing lightweight ones, to avoid overburdening the drone's processing capabilities. DroNet proposed an end-to-end navigation solution via a single neural network [12]. However, our approach leans towards a modular system as it tends to be faster and easier to debug. A faster approach is because the perception network and control network can operate at independent speeds, which is particularly advantageous for certain needs, such as the control network running at a much faster rate. Modularity also offers the benefit of providing insights into the actions of each network, rather than treating the system as a black box.

The state-of-the-art system for drone racing is Swift [4] and also follows a modular neural network approach. On the

perception side, this system runs a gate detection Convolutional Neural Network(CNN), that infers the corners of gates and the part affinity fields [13]. Following this, the gates are identified and the position of the drone on the race track is determined. This network has a U-net structure not compatible with nano drones due to its computational load. For this reason, lightweight and efficient networks, like Gatenet [8] and Dronet [12] are a better option for this project.

On the control side, Swift employs a two-layer network that maps the output of the Kalman filter to the control commands for the drone. The policy is trained using reinforcement learning in simulation [4]. During training, the policy maximizes a reward that combines progress towards the next racing gate with a perception objective that rewards keeping the next gate in the field of view of the camera. This approach is also used in the Guidance and Control Net [14], with the output of the network being the motor rpm commands directly instead of the attitude rates and thrust and no perception penalty. Deploying reinforcement learning policies on nano drones is a challenge due to their unpredictability, as there will be a large reality gap between the simulator and the real world. This reality gap is attributed to many factors namely, the sensitivity to disturbances due to the small size of nano drones, the dependency on the battery state [5], and the fact that the Crazyflie motors are brushed.

III. Hardware

To devise an optimal racing strategy for nano drones, one must first consider the inherent constraints of these small devices. We employ the CrazyFlie 2.1 as our primary model due to its widespread availability and its open-source development platform. This drone is powered by an STM32F405 processor (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash) and utilizes the nRF51822 for radio communications and power management (Cortex-M0, 32MHz, 16kb SRAM, 128kb flash). Additionally, it has 3-axis accelerometers/gyroscopes and a high-resolution pressure sensor [15]. Moreover, the CrazyFlie 2.1 is equipped with an AI-Deck that integrates a Hi-Max Gray-scale camera, with a low-power GAP8 processor.

IV. Implementation

The software stack is shown in figure 1. One can notice that the perception task is handled by the AI-Deck, where a [122x162] pixel image is captured by the Hi-Max camera at a rate of 30Hz. This image is then fed to a network that detects gates and outputs the center coordinates of the gate and the apparent size, in pixels. Following this, a relative position algorithm estimates from these parameters, the relative position from the gate to the drone. In this research, the state estimation is done with a motion capture system and an IMU. Alternatively, one could use the relative gate position and IMU but this is outside the scope of the project.

Once the states have been estimated, a Guidance and Control network [14] outputs the desired thrust and attitude rates, and will fly the drone through the gate. At the lowest level, the rates are fed to a PID controller, developed by Bitcraze, which outputs the motors' rpms.

A. Perception

1) Architecture: In order to detect a gate, a CNN-based approach is taken. Two state-of-the-art neural networks are compared in this work. The first one, GateNet [8], is a CNN with 6 convolutional layers followed by a max pooling, and at the end a fully-connected layer, illustrated in figure 2. The idea of this network would be for the convolutional layers to detect features and the fully connected layer to map these features to the pixel coordinates of the center, the width, and the height of the gate. The other network is Dronet [12], which is a variant of the ResNet-8 neural network with three residual blocks. This architecture is efficient and simplifies back-propagation. In addition, these residual blocks help combat the vanishing gradient problem [12]. The outputs of these networks are the u and v coordinates of the center of the gate, the height and width of the gate(in pixels), and a confidence value that determines if a gate is in the image or not.

2) Training: Running a neural network on the AI-Deck requires it to be lightweight, especially if it runs for drone racing purposes the latency needs to be low, leading to the need for a low-resolution input image. For this project, the networks are trained on two datasets. The first dataset consists of 5000 simulated images with the AIRR Gate [3] to compare the two network architectures and evaluate if detecting gates is possible at low resolutions[1]. The other dataset is a set of images taken with the Hi-Max camera of the gate that needs to be detected, see figure 3. This is a collection of 2000 images, of which 400 have been labeled by hand. The rest are labeled with a computer vision algorithm that detects white pixels in the image. To be automatically labelled the gate has to be on a dark background.

3) Loss Function: The network is subject to minimizing the loss function in equation 1.

$$\begin{aligned}\mathcal{L} =& \mu_{\text{center}}\hat{c}\left((u-\hat{u})^2+(v-\hat{v})^2\right)\\ &+\mu_{\text{size}}\hat{c}\left(\left(s_u-\hat{s}_u\right)^2+\left(s_v-\hat{s}_v\right)^2\right)+\mu_{\text{conf}}(c-\hat{c})^2,\end{aligned} \tag{1}$$

where $\mu_{\text{center}}=2$, $\mu_{\text{size}}=2$ and $\mu_{\text{conf}}=1$, are the weights of the center coordinate, size and confidence parts of the loss function, these are non-trainable. (u,v) and $(\hat{u},\hat{v})$ are the coordinates to the center of the gate, predicted and ground-truth respectively. s_u, s_v are the width and height of the gate in pixels.

4) Deployment: The GapFlow process, pipeline by Greenwaves technologies [16], is used to deploy the CNN on the GAP8 processor. Before the network can go through this pipeline it needs to be quantized. For this, the training-aware quantization of TensorFlow is employed. Network sparsification, to reduce the number of connections in a network, was also attempted but did not improve the performance of the network(see Additional Results). This model is then fed to the GAP NNTool, which maps the network nodes to the GAP8 computational nodes. Next, the GAP Autotiler optimizes the data movement across the memory hierarchy based on the

[1]https://github.com/open-airlab/pencilnet

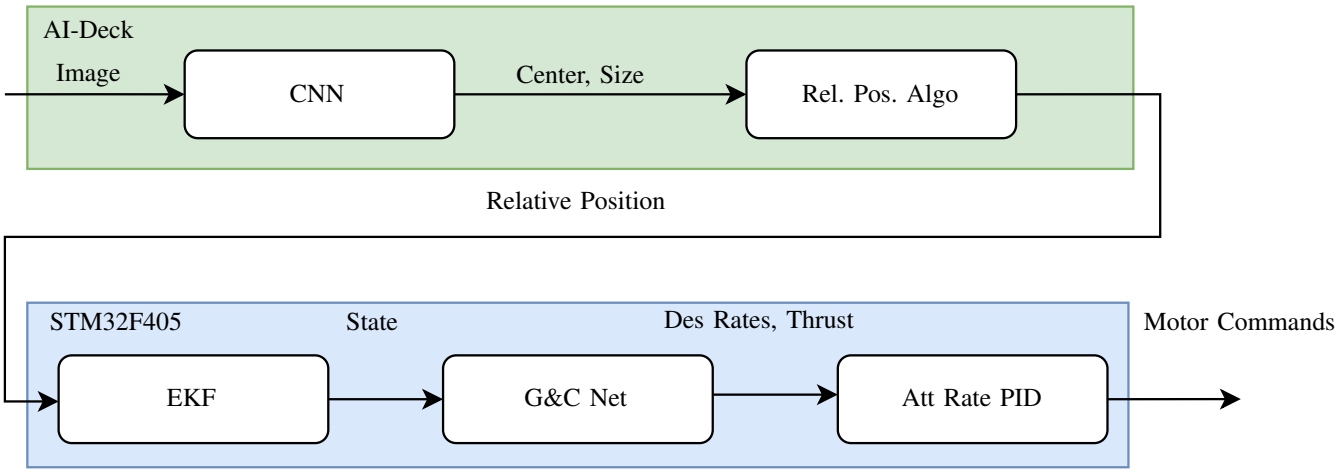


Fig. 1: Software stack for racing on the Crazyflie

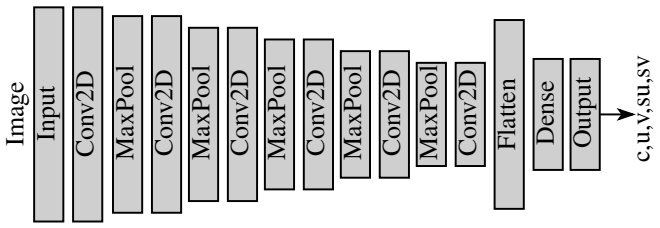


Fig. 2: Network architecture of Gatenet

Fig. 3: On the left, the test set of the simulation dataset, in green is the label and in red are results of Gatenet. On the right, two images captured by the AI-Deck camera. Again in red the output of the network, no labels are present for these images.

network and memory constraints. Finally, this generated code can be dockerized and flashed to the AI-Deck with the help of the crazyradio.

B. Relative Position

The relative position from the gate to the drone is estimated using the center coordinates and size of the gate, which are computed by the gate detection network. The first step is to calculate the angle, ζ, between the gate center and the center of the camera, shown in equation 2. Where u and v are the center coordinates of the gate and camera, in pixels. While, f is the focal length.

$$\zeta = \frac{v_{\text{gate}} - v_{\text{cam}}}{f_y}, \zeta_y = \frac{u_{\text{gate}} - u_{\text{cam}}}{f_x} \tag{2}$$

Following this, the distance to the camera is calculated according to equation 3, where s_{real} is the size of the gate, assumed to be known, in meters and s_{net} is the size of the gate in pixels, which is the output of the gate detection network. The distance can be calculated using s_u or s_v values according to which measurement is larger. The camera focal length and center are determined experimentally.

$$d = s_{\text{real}_x} \times \left(\frac{f_x}{s_{\text{net}_u}} \right) \tag{3}$$

Finally, x, y, and z relative position from the center of the gate to the drone are extracted using equations 4, 5 and 6. The coordinate system is z up with the origin located at the center of the gate, the x-axis is aligned with the direction the gate needs to be crossed. Thus, the drone is always behind the gate. The major drawback of this approach is that it assumes the relative yaw between the gate and the drone to be zero.

$$x = -d \tag{4}$$
$$y = -\zeta_x \times d \tag{5}$$
$$z = -\zeta_y \times d \tag{6}$$

C. Control

Once the relative position has been determined, a feed-forward network estimates the desired attitude rates and thrust, then a PID controller yields the motor rpms'. To achieve this we train the model in simulation using reinforcement learning.

The pipeline begins with the drone being modeled to have a good simulator, next the architecture of the network was decided. Following this, the network was trained in simulation leveraging the Markov Decision Process(MDP) framework. Finally, the network is deployed on the flight controller. To fly the control network an imaginary race is set up, with four gates in a 4×3m rectangle. The goal is to fly this path as quickly as possible.

1) Quadcopter Model: The controller's response to commands are assumed to be a first-order delay system. The model encompasses kinematic equations and a linear drag model, the states and inputs are:

$$\mathbf{x} = [\mathbf{p}, \mathbf{v}, \boldsymbol{\lambda}, \boldsymbol{\Omega}, T]^T, \quad \mathbf{u} = [\boldsymbol{\Omega}_{\text{cmd}}, T_{\text{cmd}}]^T$$

Where $\mathbf{p}$ are the positions of the drone, $\mathbf{v}$ are the velocities, λ are the Euler angles, $\boldsymbol{\Omega}$ are the angular rates, and T is the thrust. The equations governing motion are:

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{7}$$
$$\dot{\mathbf{v}} = g\mathbf{e}_3 + R(\boldsymbol{\lambda})\mathbf{F}, \tag{8}$$
$$\dot{\boldsymbol{\lambda}} = Q(\boldsymbol{\lambda})\boldsymbol{\Omega}, \tag{9}$$
$$\dot{\boldsymbol{\Omega}} = (\boldsymbol{\Omega}_{\text{cmd}} - \boldsymbol{\Omega})/\tau_{\Omega}, \tag{10}$$
$$\dot{T} = (T_{\text{cmd}} - T)/\tau_T, \tag{11}$$

From flight data we computed τ_p and τ_q to be 0.0382. τ_r to be 1.1113, and τ_T to be 0.0565,. Moreover, the specific force $\mathbf{F}$ is formulated as $\mathbf{F} = [-d_x v_x^B, -d_y v_y^B, -T]^T$. From empirical data, we derived d_x to be 0.34 and d_y to be 0.43.

To optimize performance, we set defined boundaries for thrust and attitude rates, tailored to the most stable and rapid flight achievable in our system:

$$p_{\text{cmd}}, q_{\text{cmd}} \in [-0.2, 0.2], \quad T_{\text{cmd}} \in [7.8, 10.6], , r_{\text{cmd}} \in [-0.6, 0.6]$$

2) Architecture: The policy network processes 13 inputs, which include states of the drone relative to the gate, denoted by superscripts g, attitude rates, and thrust:

$$\mathbf{x}_{\text{in}} = [\mathbf{p}^{g_i}, \mathbf{v}^{g_i}, \boldsymbol{\lambda}^{g_i}, \boldsymbol{\Omega}, T]^T$$

The network outputs four control signals: thrust and body rate commands. Distributed from -1 to 1 to simplify the network learning. The architecture consists of three general matrix multiply(GeMM) layers, with a ReLu activation function, with the same amount of neurons, see in figure 4.

3) Training: In order to train the network using reinforcement learning, we transform our dynamic model into a discrete-time MDP. This framework defines the set of states the environment can be in, the set of actions the agent(drone) can take, the reward for every given action, and the probabilities of moving from one state to the next after an action. Initial states are uniformly sampled from specified intervals, as described in equation 12. The agent starts at one of these states and employs the network to choose an action with a certain probability and receiving a certain reward, r_t.

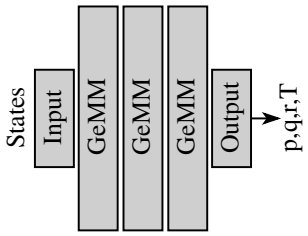


Fig. 4: Network architecture of the Guidance and Control network

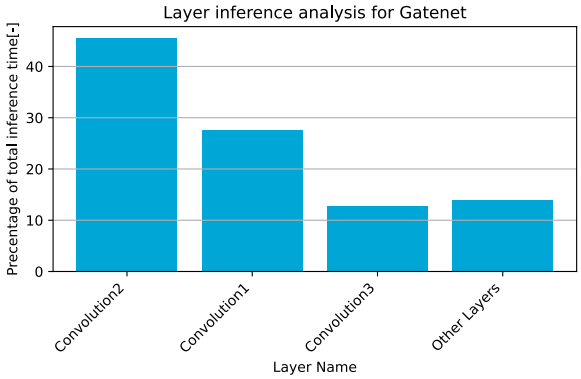


Fig. 5: Inference analysis of the layers of GateNet as a percentage of the total inference time. Inference is run on the simulation dataset

The objective is to change the parameters (w) of a neural network policy to maximize expected return $\max_w \mathbf{E}_{\pi_w}[\sum_{t=0}^{\infty} \gamma^t r_t])$.

$$\begin{array}{lll} x \in [-\frac{1}{2}, \frac{1}{2}] + x_s & y \in [-\frac{1}{2}, \frac{1}{2}] + y_s & z \in [-\frac{1}{2}, \frac{1}{2}] + z_s \\ v_x \in [-\frac{1}{2}, \frac{1}{2}] & v_y \in [-\frac{1}{2}, \frac{1}{2}] & v_z \in [-\frac{1}{2}, \frac{1}{2}] \\ \phi \in [-\frac{2\pi}{9}, \frac{2\pi}{9}] & \theta \in [-\frac{2\pi}{9}, \frac{2\pi}{9}] & \psi \in [-\pi, \pi] \\ p \in [-1, 1] & q \in [-1, 1] & r \in [-1, 1] \\ T \in [9.8, 10.1] & & \end{array} \tag{12}$$

The drone's goal is to complete an oval path, moving through each gate as rapidly as possible. The performance is evaluated using a reward function 13, as implemented by Ferede et al. [14] with the additional yaw penalty.

$$r(k) = \begin{cases} ||\mathbf{p}_k - \mathbf{p}_{g_k}|| - ||\mathbf{p}_{k-1} - \mathbf{p}_{g_k}|| & \\ -(\psi_k - \psi_{g_k})^2 & \text{if } r > 30 \\ +10 - 10||\mathbf{p}_k - \mathbf{p}_{g_k}||, & \text{if gate passed} \\ -10, & \text{if collision} \end{cases} \tag{13}$$

In this function, $\mathbf{p}_{g_k}$ indicates the center position of the current target gate. $\mathbf{p}_k$ and $\mathbf{p}_{k-1}$ represent the drone's present and preceding positions, respectively. The yaw is also taken into account as we always want the drone to face the gate, to ensure that the camera sees the gate. This reward becomes active only once the network manages to fly through 3 gates, making the

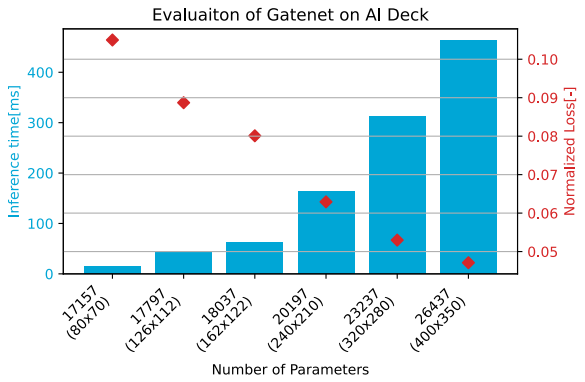


Fig. 6: Trade-off between inference time on the AI-Deck and the loss of networks with gatenet architecture with different input image sizes. Trained on the self-generated dataset.

training more effective. Successful gate passage occurs when the drone crosses the specified 1x1 meter boundary of the gate. Collisions happen either when the drone touches the ground or if it crosses outside the designated boundary of the gate.

4) Deployment: For optimal performance, the attitude rates need to be supplied at approximately $150Hz$, to receive up-to-date information but not overload the processor. To achieve this speed on an STM32F405 processor, the network needs to be lightweight. For this reason, after the training carried out with Stable Baselines Proximal Policy Optimization [17], an ONNX model is extracted and quantized with dynamic quantization, before being converted to C code[2].

V. Results

In this work, we evaluate perception and control separately. The perception networks are tailored for the AI-Deck, while the control network is deployed on the STM32F405. All flight tests were conducted in the CyberZoo, a 10x10x7 m flight arena at TU Delft's Aerospace Engineering faculty, equipped with an OptiTrack motion capture system for real-time position. The autopilot used was the Bitcraze software development kit for the Crazyflie.

A. Perception

1) Architecture Selection: Detecting gates from images of small resolution is a complex task for this reason two different neural network architectures are investigated and compared. Table I compares the Dronet [12] and Gatenet [8] architectures in terms of FPS on an Intel-i5 CPU with no multi-threading. We compare the loss according to the loss function, shown in equation 1, the error in the center coordinated ($\mathbf{E_c}$), and errors in width and height($\mathbf{E_{sx}}$ and $\mathbf{E_{sy}}$) according to equations 14 and 15 respectively.

$$E_c = |u - \hat{u}| + |v - \hat{v}| \tag{14}$$

$$E_s = |s - \hat{s}| \tag{15}$$

[2]https://github.com/kraiskil/onnx2c

The experiment found that Dronet consistently surpassed Gatenet across all error terms, while also having a lower latency. It is important to note that in the simulation dataset, all the images contain a gate therefore the confidence error is not taken into account, but can be used as a framework for further work.

Upon detailed examination, the disparity in the error metrics between the two architectures was relatively marginal. Specifically, the relative error in determining the center pixel location was 2 pixels, while the error in estimating the size of the gate was approximately 0.7 pixels. These results suggest a closer performance level between the two models than initially indicated by the loss term.

However, a critical limitation of the Dronet architecture arises from its structural complexity. Dronet's architecture cannot be quantized using either TensorFlow or the GapSDK routines. This constraint makes Dronet impractical for implementation on the AI-Deck platform. Given these technical limitations and the necessity for a balance between performance and deployability, Gatenet is further analyzed in detail in this work.

Network	FPS	Loss	$\mathbf{E_c}$[pix]	$\mathbf{E_{su}}$[pix]	$\mathbf{E_{sv}}$ [pix]
DroNet	764	83.64	8.35	1.50	0.92
GateNet	528	127.19	10.97	2.16	1.60

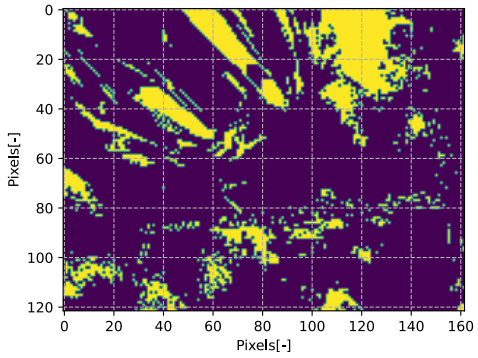
TABLE I: Comparison between DroNet and GateNet in terms of errors and inference times for images of size $[122 \times 162]$ and batch size 32. Trained and tested on the simulation dataset

2) Layer Inference Analysis: The performance of Gatenet was benchmarked on an Intel Core i5 processor, where it achieved a frame rate of 528 frames per second (FPS). Given the high-speed demands of drone racing, it is imperative to maximize the network's processing speed. The GAP8 processor, featuring a RISC-V architecture, shows architectural parallels to the x86 architecture of the Intel i5. This similarity facilitates the transferability of inference analysis between these two platforms. In Figure 5, we present a detailed breakdown of the computation time required for each layer in the Gatenet architecture, expressed as a percentage of the total inference time.

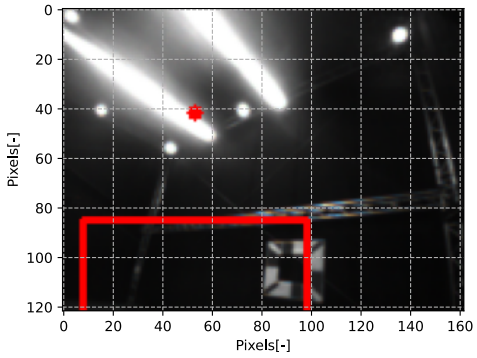
The most important observation from this analysis is that, out of the 15 layers, the first three convolutional layers account for approximately 75% of the total inference time. This significant proportion shows that these layers are the primary bottlenecks in the network's computation pipeline. Consequently, reducing the resolution of the input images leads to a substantial decrease in latency.

Another observation is that the second convolution is slower, primarily because of the significantly higher number of multiplications needed due to the increased number of channels in the input and the increased size of filters, despite the smaller height and width of the input layer.

3) Parameter Tuning: In the pursuit of optimizing the Gatenet architecture for deployment on the AI-Deck, a series of experiments were conducted to determine the most effective input image size. The primary objective was to establish a balance between inference time and accuracy, ensuring that the

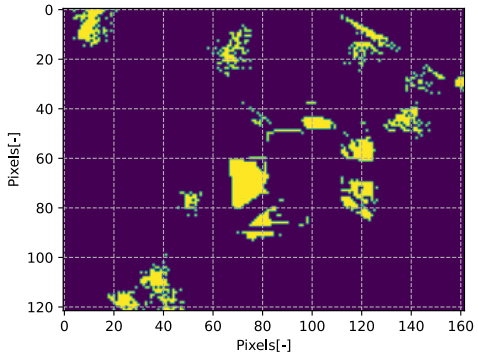


(a) Mask of the LIME algorithm, the yellow pixels are the most sensitive

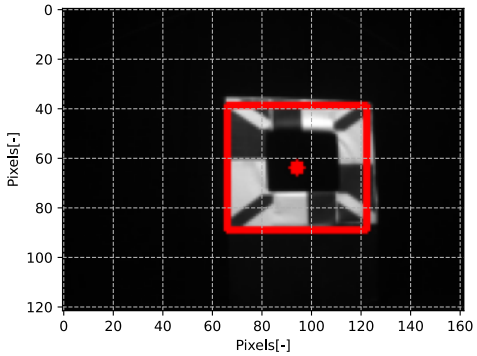


(b) Prediction of Gatenet on the image given to LIME algorithm

Fig. 7: LIME on testset image with light in the camera view



(a) Mask of the LIME algorithm, the yellow pixels are the most sensitive



(b) Prediction of Gatenet on the image given to LIME algorithm

Fig. 8: LIME on standard testset image

network maintains a high performance while operating within acceptable time constraints. The results of these experiments are shown in Figure 6. Normalized loss is the loss from equation 1 with the center and size prediction divided by the height or width of the input image.

The experimental data shows that the inference time increases with an increase in image size. For instance, with an input dimension of $[80 \times 70]$ pixels, the network ran at an inference time of 15 ms, corresponding to a frequency of approximately 66 Hz, and recorded a loss of 0.11. While a $[400 \times 350]$ image runs at $463ms[2Hz]$. This observation highlights the direct impact of image size on the computational efficiency and accuracy of the network.

After evaluating various configurations, the $[162 \times 122]$ pixel version of Gatenet was selected. This decision was informed by the requirement to keep the vision system's update rate within a minimum threshold of approximately 10 Hz.

4) LIME Algorithm: In order to better understand the results and limitations of the network. The Local Interpretable Model-agnostic Explanations(LIME) algorithm is employed [18].

LIME operates by first perturbing the input data and generating a new dataset consisting of these perturbed instances along with the corresponding predictions of the network [18]. For an individual prediction to be explained, LIME focuses on this local neighborhood generated by perturbation. It then learns a simple, interpretable model, such as a linear model, which is trained to approximate the predictions of the complex model as closely as possible within this local space.

The learned interpretable model is used to identify the importance of each feature for the prediction of the instance being explained. This is achieved by examining the weights of the linear model that lead to the decision. The features that contribute most significantly to the prediction are considered to be key explanatory factors.

The output of this algorithm is shown in figures 7 and 8. Focusing on figure 7, one can immediately notice that the network gives importance to the white sections of the gate, namely the bottom left corner and the top left corner. Apart from some noise on the side of the images, the most sensitive pixels are around the gate. Figure 7 illustrates the network's deficiencies. It depicts the white lights on the ceiling of the test environment, which correspond to white pixels that are absent

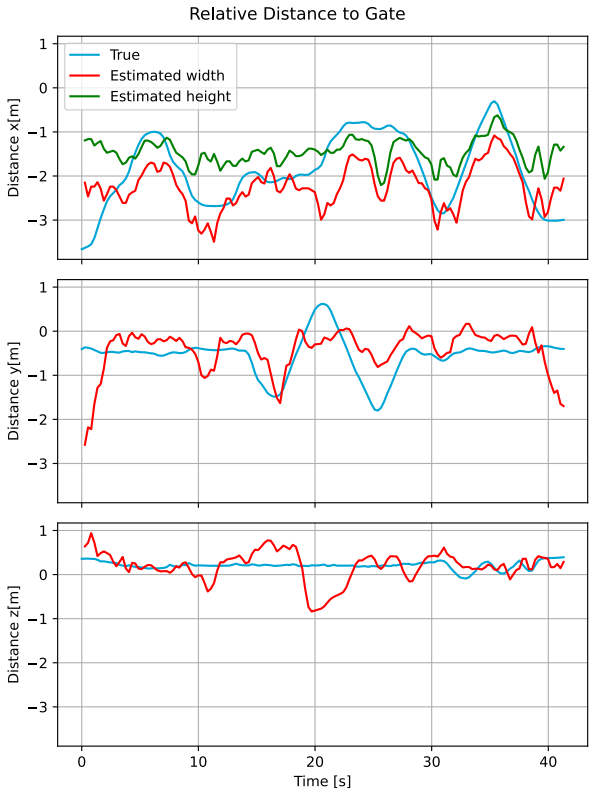


Fig. 9: Comparison between the position of the drone relative to the gate measured with a motion capture system and estimated by the CNN and relative position pipeline, estimate width is calculated with the dimension of the width and the height with the dimensions of the height. The coordinate system is z up with the origin located at the center of the gate, the x-axis is aligned with the direction the gate needs to be crossed. Thus, the drone is always behind the gate.

in the training dataset. Consequently, the network exhibits a pronounced bias towards regions with more white pixels. Additionally, the network incorrectly infers the dimensions of the gate, estimating it to be negative.

5) Relative Position: Figure 3 shows the AI-Deck running Gatenet onboard can recognize a gate and estimate its center and size in pixels. Figure 9 compares the output of the relative position onboard running onboard against the true distance from the gate to the drone over a 40-second period.

It can clearly be seen that the actual and predicted values for x are closely aligned, indicating that the network reliably predicts the gate's dimensions and any size variation. However, there is a notable discrepancy in the estimated distance when the drone is situated more than 3 meters from the gate, particularly at the start and end of the flight. This error is linked to the CNN's tendency to overreact to white pixels. Beyond the 3-meter mark, the ceiling lights in the testing environment come into the camera's view, compromising the accuracy of the predictions.

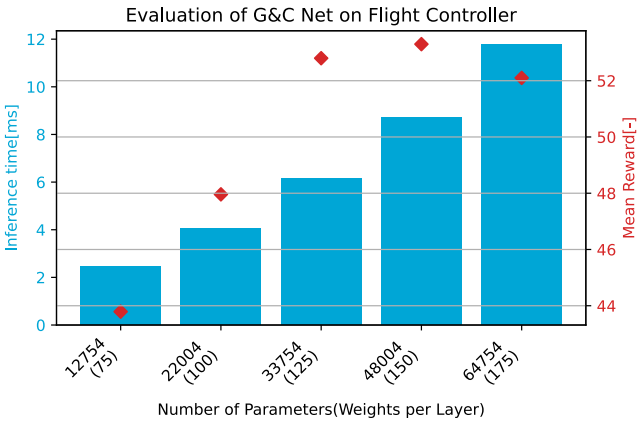


Fig. 10: Trade off between the inference time on the Crazyflie flight controller and the reward of different size networks

B. Control

1) Parameter Tuning: Once again, for the application of autonomous drone racing a faster network will result in a faster drone. For this reason, a series of experiments were conducted in which the number of neurons per layer of the network was changed, thus changing the number of parameters. These different networks were trained and the inference time on the Crazyflie flight controller was recorded.

The findings, illustrated in Figure 10, reveal a direct correlation between the network's size and its inference time, which ranged from $2.4ms$ to $11.7ms$. A particularly interesting aspect of these results is the observed plateau in the mean reward, irrespective of the network's increasing size. Despite increasing the network dimensions, the mean reward appeared to stabilize around 50. This figure suggests that, under the conditions described in the previous section, the drone averaged passing through five gates over a 12-second duration. Interestingly, while an increase in network size made the training faster, it did not increase the drone's gate navigation performance.

The Crazyflie's attitude rate PID operates at $500Hz$. For optimal performance, the attitude rates need to be supplied at approximately $150Hz$. This requirement positions the network configuration with 125 neurons as the most effective.

2) Performance Analysis: Tests were carried out on both simulated rollouts and actual flight experiments. An analysis of the real flights, as depicted in Figure 11, reveals that the overall quality of flight control is suboptimal. This deficiency is attributed to two principal factors. Firstly, there is a notable discrepancy in the drone's yaw control, as illustrated in Figure 12. The inaccuracy in modeling the drone's yaw significantly impairs the network's ability to manage this aspect of flight control, a limitation that becomes evident in the observed small looping of the drone's flight trajectory.

Secondly, the instability can be linked to the thrust model of the Crazyflie. While the thrust appears to be accurately modeled as seen in Figure 12, the underlying issue arises from how the network directly manipulates the thrust. Specifically, the network's output in Gs is transformed into a thrust command

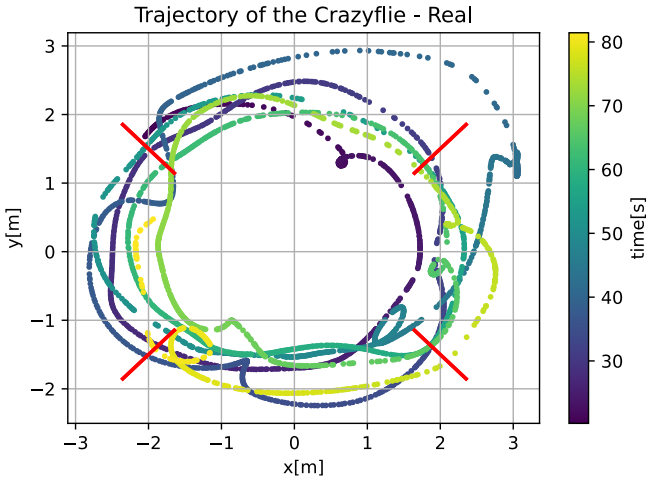


Fig. 11: Trajectory for the real flight of the Crazyflie.

via multiplication with a constant. This constant, although empirically derived, shows a high degree of sensitivity to the drone's battery state and voltage. Additionally, our model does not account for the reduced efficiency of the propellers at higher velocities, a factor that further compromises the drone's flight stability and control precision. This highlights the complex interplay between modeling accuracy and real-world control dynamics, proving the need for more sophisticated models.

Analysing the simulated flight, as depicted in Figure 13. A notable observation from the simulation data is the smoother trajectory achieved, due to the more precise control over thrust and yaw within the simulation environment. However, despite the smoother trajectory, some inconsistencies are present, especially in the first lap. These errors are less visible when the network is trained with higher limits on the commands, the reason for this is unknown and should be addressed in further research.

Another insight emerged when examining the time taken to complete the flight paths: the simulated flight accomplished five laps in 18 seconds, in contrast to the 80 seconds required for the real flight. This significant discrepancy underscores the reality gap. Specifically, in real-world scenarios, the flight is considerably slowed down by the need for numerous correction maneuvers.

Figure 12 shows the command and response of the Crazyflie during the flight. It is clear that the modeling of the drone is not very accurate, the thrust is acceptable, in this portion of the flight. While, the roll and pitch rate overshoot. Another important aspect is the yaw, as the model does not follow the behavior of the drone. Moreover, the sensor heavily undershoots the command especially in positive rotations. It is important to note that a bias was identified of $35deg/s$ between network command and response signifying an error in the PID controller, this was corrected in the flight tests.

VI. Conclusion

We showed that state-of-the-art neural networks for drone racing can be deployed on nano drones, despite their limited

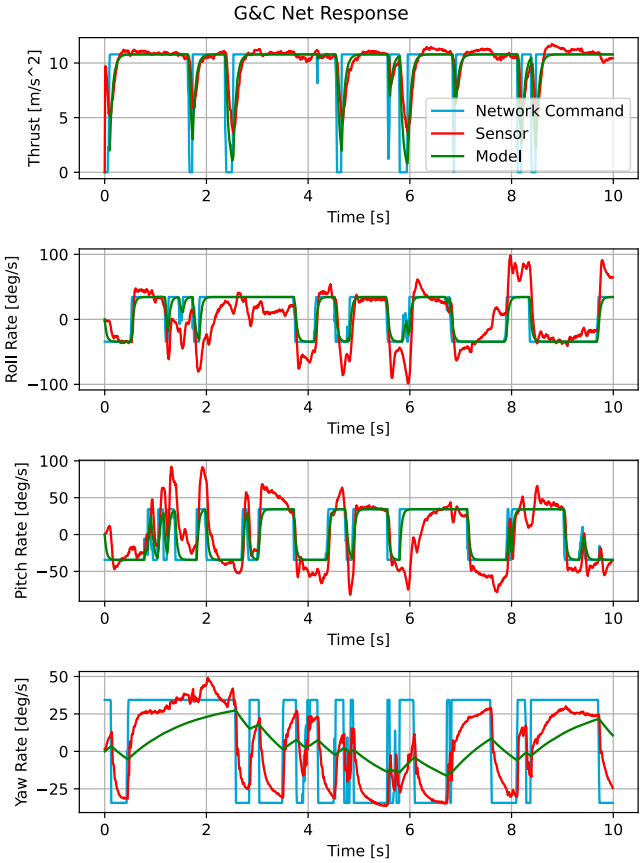


Fig. 12: Comparison between the network commands and the response of the drone over the first 10 seconds of flight.

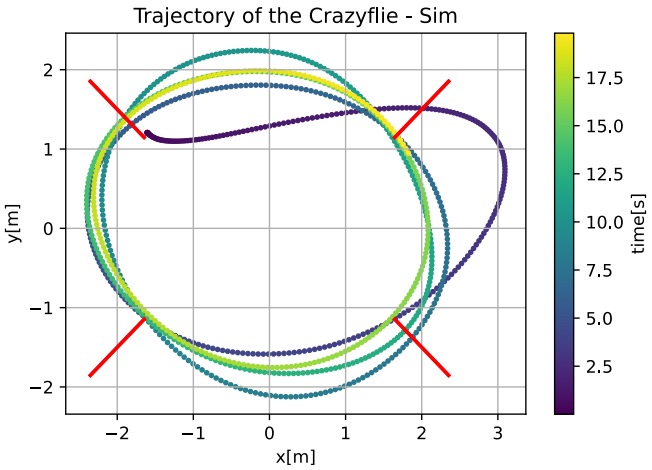


Fig. 13: Simulated trajectory of the Crazyflie.

processing power. Our approach is divided into perception and control. The former consists of a lightweight CNN with a gatenet-like architecture [8]. We have tuned and deployed this network on a GAP8 processor, achieving speeds of $16Hz$. This network outputs the size and center location of the gate which are then fed to a relative position algorithm. A second network trained with reinforcement learning is in charge of guidance and control for the drone. This network was deployed on the STM32F405 and, once again after having tuned the number of parameters, the inference time is 6ms. Proving, that it is possible to run state-of-the-art neural networks on small, CPU-constrained devices.

This work is a proof of concept for nano drone racing, further work on the perception part of the project should focus on one of the biggest limitations of the CNN, namely the sensitivity to white pixels. This issue arises from a dataset that mostly represents the gate on a black background, to facilitate the self-labeling algorithm. Moreover, deploying Dronet [12] on the AI-Deck will be beneficial considering it outperformed Gatenet in terms of accuracy and inference time. Another aspect that should be researched could be to include the skew of the gate in the prediction of the network to be able to extract the relative yaw between the gate and the drone.

With regards to the control side of the project, further steps could be focused on a better model of the drone to fix the attitude rates responses. To fix the thrust two approaches can be carried out, the first one is to modify the model such that during the training phase the maximum thrust varies thus leading to a network that can adapt to changes in maximum thrust. The second approach could be to implement a PID controller on the thrust command, similar to what is done for the attitude rates. Another interesting path to take would be to improve the attitude rate PID controller of the Crazyflie or substitute it with an INDI controller.

REFERENCES

[1] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, p. 604–609, Dec. 2020.

[2] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, p. 350–354, Nov. 2019.

[3] C. De Wagter, F. Paredes-Vallé, N. Sheth, and G. de Croon, "The sensing, state-estimation, and control behind the winning entry to the 2019 artificial intelligence robotic racing competition," *Field Robotics*, vol. 2, no. 1, p. 1263–1290, Mar. 2022.

[4] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 79767976, p. 982–987, Aug. 2023.

[5] L. Lamberti, V. Niculescu, M. Barciś, L. Bellone, E. Natalizio, L. Benini, and D. Palossi, "Tiny-pulp-dronets: Squeezing neural networks for faster and lighter inference on multi-tasking autonomous nano-drones," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2022, p. 287–290.

[6] M. J. Anderson, J. G. Sullivan, J. L. Talley, K. M. Brink, S. B. Fuller, and T. L. Daniel, "The "smellicopter," a bio-hybrid odor localizing nano air vehicle," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6077–6082.

[7] D. Palossi, N. Zimmerman, A. Burrello, F. Conti, H. Müller, L. M. Gambardella, L. Benini, A. Giusti, and J. Guzzi, "Fully onboard ai-powered human-drone pose estimation on ultra-low power autonomous flying nano-uavs," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[8] H. X. Pham, I. Bozcan, A. Sarabakha, S. Haddadin, and E. Kayacan, "Gatenet: An efficient deep neural network architecture for gate perception using fish-eye camera in autonomous drone racing," pp. 4176–4183, 2021.

[9] D. Scaramuzza and Z. Zhang, "Visual-inertial odometry of aerial robots," no. arXiv:1906.03289, Jun. 2019, arXiv:1906.03289 [cs]. [Online]. Available: http://arxiv.org/abs/1906.03289

[10] J. A. Cocoma-Ortega and J. Martínez-Carranza, "Towards high-speed localisation for autonomous drone racing," in *Advances in Soft Computing*, ser. Lecture Notes in Computer Science, L. Martínez-Villaseñor, I. Batyrshin, and A. Marín-Hernández, Eds. Cham: Springer International Publishing, 2019, p. 740–751.

[11] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, "Model predictive control for micro aerial vehicles: A survey," no. arXiv:2011.11104, Nov. 2020, arXiv:2011.11104 [cs]. [Online]. Available: http://arxiv.org/abs/2011.11104

[12] A. Loquercio, A. I. Maqueda, C. R. del Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, p. 1088–1095, 4 2018.

[13] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "Alphapilot: autonomous drone racing," *Autonomous Robots*, vol. 46, no. 1, p. 307–320, Jan. 2022, company: Springer Distributor: Springer Institution: Springer Label: Springer number: 1 publisher: Springer US Citiation Key: alphapilot.

[14] R. Ferede, C. De Wagter, G. de Croon, and D. Izzo, "End-to-end reinforcement learning for time-optimal quadcopter flight," in *ICRA 2024*, 2023.

[15] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017, pp. 37–42.

[16] "GAPflow," https://greenwaves-technologies.com/gapflow/, accessed: 2023-11-21.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," *CoRR*, vol. abs/1602.04938, 2016. [Online]. Available: http://arxiv.org/abs/1602.04938