2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

N-MPC for Deep Neural Network-Based Collision Avoidance exploiting Depth Images

Martin Jacquet[1], Kostas Alexis[1]

Abstract— **This paper introduces a Nonlinear Model Predictive Control (N-MPC) framework exploiting a Deep Neural Network for processing onboard-captured depth images for collision avoidance in trajectory-tracking tasks with UAVs. The network is trained on simulated depth images to output a collision score for queried 3D points within the sensor field of view. Then, this network is translated into an algebraic symbolic equation and included in the N-MPC, explicitly constraining predicted positions to be collision-free throughout the receding horizon. The N-MPC achieves real time control of a UAV with a control frequency of 100Hz. The proposed framework is validated through statistical analysis of the collision classifier network, as well as Gazebo simulations and real experiments to assess the resulting capabilities of the N-MPC to effectively avoid collisions in cluttered environments. The associated code is released open-source.**

I. Introduction

Aerial Robots (ARs) are increasingly used in a large range of autonomous applications, from aerial monitoring or exploration to working in high-risk places or human-denied areas, or in tasks such as search-and-rescue [1], subterranean exploration [2] and indoor building inspection [3]. Furthermore, the increasing efficiency and decreasing weight of the available sensors and computation units allowed the deployment of recent efficient computer vision algorithms on UAVs [4], [5]. However, full autonomy of ARs in unknown and possibly cluttered environments remains a challenging task, as localization and mapping – relying only on onboard sensors – is subject to significant noise and drift, especially in visually degraded environments [6]. Although there exist active sensors, such as lidars, allowing precise high-density mapping which can be exploited by a collision-avoidance planner, those are low-frequency and heavy sensors that are not suitable for fast flights. Furthermore, the high computational requirements of dense-map planning algorithms are subsequently limiting the velocity of ARs. This issue becomes prominent in time-critical applications, and greatly reduces the distance coverage for a given battery time, e.g. in exploration tasks.

Thus, recent works [7]–[14] are taking a different approach by relying purely on sensor data and local estimates for short-term collision avoidance, tackled at the level of the controller. The main challenge in such methods lies in the high dimension of sensor data, rendering classical approaches (e.g., [7]) difficult to implement. Instead, some of the aforementioned works have proven that Neural Networks (NNs) are efficient tools to deal with such large input spaces. In this context, the favored sensors are depth images, which are both easy to embed onboard and easier to simulate than classical RGB cameras. Using such data allows to train on large batches of simulated images with a relatively small sim-to-real gap. Some recent works are also investigating learning navigation from RGB images [10], [11].

Common approaches to such sensor-based navigation policies are Imitation Learning and Reinforcement Learning (RL). The former makes use of a privileged policy [8]–[10] (e.g. exploiting map knowledge) that is imitated by a NN-based controller accessing only sensor measurements. Such approaches are however limited by the availability of such privileged policies, and do not generalize well to unknown situations. RL methods [11], [12] on the other hand are attracting more and more attention with the surge of efficient simulation environments, and recent demonstration of performances, e.g. in drone racing [15]. RL provides efficient end-to-end control schemes and is successfully employed for sensor-based collision-free navigation [11]–[13]. Recent evidence [16] show that under some assumptions on the reward function, such methods also provides tools for safety certification, through Control Barrier Function (CBF), paving the path for further research on safety-oriented RL.

In the scope of collision avoidance, NNs have also been used in combination with classical planning or control methods. Contrary to end-to-end methods, it enables a more transparent correspondence with the modular approach which has largely governed robotics research over the past years. Corollary, it allows more control on the framework by allowing evaluation the performances of the individual blocks. In [17], a NN exploits lidar data to synthesize a CBF that certifies safety of the system in the current observable environment, from which safe commands are derived using classical control. In [14], a NN is used to predict collision scores of some motion primitives based on partial current state estimates, in a receding horizon fashion. The predictive aspect is thus handled by the NN which approximates collision rollouts. However, this method is limited by the choice of sampling of trajectories, while predictive optimal local planners or controllers, such as Nonlinear Model Predictive Control (N-MPC), provide more flexibility.

However, N-MPC has not been used in combination with NNs for sensor-based navigation, but mainly for learning the dynamics residual to reduce the imprecision of the models. To this end, recent paradigms have been proposed [18]–[20] to integrate NN in N-MPC schemes. A first approach is to leverage NNs to guide sampling-based MPC [18]. On the other hand, in [20], the authors propose a so-called Neural

[1]Autonomous Robots Lab, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, martin.jacquet@ntnu.no, konstantinos.alexis@ntnu.no
This work was partially supported by the European Commission Horizon project DIGIFOREST (EC 101070405).

MPC, a MPC using deep learning models in its prediction step. The NN is directly embedded into the optimal problem as an algebraic symbolic equation, enabling gradient-based optimization.

In this work, we propose to combine the local planning aspects of N-MPC schemes with deep NN to process input data, achieving sensor-based collision avoidance in real time. A deep NN is designed for collision prediction using depth images. This NN is queried for position states and outputs a collision score. Using the same approach as [20], this NN is integrated into the NonLinear Programming (NLP) equations, constraining predicted positions to remain collision-free. The structure of the NN is designed such that the size of the matrices defining the symbolic neural prediction constraint is maintained small, enabling fast optimization, and consequently real time control.

The paper is organized as follows: first, the modeling of the AR is formally introduced. Then, the deep NN architecture is presented in Sec. III and the training methodology is defined, leading to the definition of the collision-aware N-MPC in Sec. IV. Finally, the method is evaluated both in simulations and real experiments in Sec. V, before concluding.

II. Modeling

We define the world inertial frame $\mathcal{F}_W$, with its origin O_W and its axes $\mathbf{x}_W, \mathbf{y}_W, \mathbf{z}_W$. Following the same convention, the body frame of a AR and the depth camera frame are respectively denoted $\mathcal{F}_B$ and $\mathcal{F}_C$.

The AR is defined as a rigid body centered in O_B, and actuated by typically 4 or 6 co-planar propellers. Its position with respect of (w.r.t.) the $\mathcal{F}_W$ is denoted by ${}^W\mathbf{p}_B$ and the rotation matrix from $\mathcal{F}_B$ to $\mathcal{F}_W$ is denoted by ${}^W\mathbf{R}_B$; and similarly for all the other frame pairs. The unit quaternion representation of the rotation ${}^W\mathbf{R}_B$ is denoted ${}^W\mathbf{q}_B$.

The AR is assumed to embed a front-facing depth camera, rigidly attached such that ${}^B\mathbf{p}_C$ and ${}^B\mathbf{R}_C$ are constant and known. Its Field of View (FoV), denoted $\mathbb{F}$, is a pyramidal volume described by two angles α_V and α_H and a height $d_{\max}$, respectively describing the halved vertical and horizontal angular apertures and maximum sensing depth. Its principal axis is aligned with $\mathbf{z}_C$. This camera provides, at a given frequency f_C, a depth image $I(t)$, rasterized in $n_V \times n_H$ pixels, whose scalar values the depth of the closest objects in the corresponding angular sector of $\mathbb{F}$. Pixel depth values are normalized by $d_{\max}$ such that they range in $(0,\ 1]$.

The system is expected to handle obstacles through the images captured by the depth camera. Therefore, the motion of the AR must occur within the camera FoV to enable obstacle avoidance, implying that only forward motion with pitching and yawing are allowed. The dynamics of the AR are accordingly restricted to those of a non-holonomic system.

The system state is described by the vector

$$\mathbf{x} = [\mathbf{p}^\top\ \mathbf{q}^\top\ v_x]^\top \in \mathbb{R}^8, \tag{1}$$

where $\mathbf{p} = {}^W\mathbf{p}_B$, $\mathbf{q} = {}^W\mathbf{q}_B$, v_x is the forward velocity of O_B, expressed in $\mathcal{F}_B$. The system input variables are

$$\mathbf{u} = [a_x\ \omega_y\ \omega_z]^\top \in \mathbb{R}^3, \tag{2}$$

where ω_y and ω_z are pitching and yawing rates of $\mathcal{F}_B$ w.r.t. $\mathcal{F}_W$, expressed in $\mathcal{F}_B$, and a_x is the forward acceleration of of O_B, also expressed in $\mathcal{F}_B$.

Accordingly, the system kinematics and dynamics are defined by

$$\dot{\mathbf{p}} = {}^W\mathbf{R}_B \begin{bmatrix}0\ 0\ v_x\end{bmatrix}^\top, \tag{3a}$$

$$\dot{\mathbf{q}} = \frac{1}{2}\ \mathbf{q} \otimes \begin{bmatrix}0\ 0\ \omega_y\ \omega_z\end{bmatrix}^\top, \tag{3b}$$

$$\dot{v_x} = a_x, \tag{3c}$$

where $\otimes$ denotes the Hamilton product of two quaternions.

III. Deep Neural Collision Predictor

Given a depth image I and a 3D point $\mathbf{a}$, there exists a mapping

$$\begin{aligned} g\ :\ \mathbb{I} \times \mathbb{F}\ &\to \{0,\ 1\} \\ (I, \mathbf{a})\ &\mapsto\ \ c \end{aligned} \tag{4}$$

where $\mathbb{I} = (0,\ 1]^{(n_V \times n_H)}$ is the depth image space, and c is a Boolean value describing whether $\mathbf{a}$ is in collision with an obstacle in I. Note that points that fall behind obstacles are not visible from the depth camera, and thus are considered to be in collision. Intuitively, this implies that any point that is not directly visible potentially collides with an obstacle, and therefore is conservatively classified as being in collision. This assumption is also used to extend the domain of definition of g to $\mathbb{I} \times \mathbb{R}^3$ with $g(I, \mathbf{a}) = 1,\ \forall \mathbf{a} \notin \mathbb{F}$.

Such function g is discontinuous and challenging to write in closed form, therefore it does not allow any gradient-based optimization in order to let a N-MPC avoid collisions throughout its receding horizon.

Instead, a continuous parametric approximation of g can be defined as

$$\begin{aligned} g_{\boldsymbol{\theta}}\ :\ \mathbb{I} \times \mathbb{R}^3\ &\to [0,\ 1] \\ (I, \mathbf{a})\ &\mapsto\ \ \hat{c} \end{aligned} \tag{5}$$

where $\boldsymbol{\theta}$ is a set of parameters and $\hat{c}$ is an approximate collision score such that

$$\forall (I, \mathbf{a}), \quad \begin{cases} \hat{c} \approx 1 & \text{if } c = 1, \\ \hat{c} \approx 0 & \text{if } c = 0. \end{cases} \tag{6}$$

We note that the method itself is not restricted to using depth images. Other depth-based sensors such as lidars could be employed similarly.

A. Neural Network Architecture

Our objective is to design a deep NN that learns an accurate approximation $g_{\boldsymbol{\theta}}$ of g.

This network is designed after Variational Encoder-Decoder architectures, depicted in Fig. 1. The variational encoder is a Convolutional Neural Network (CNN) that processes the input image I into a latent representation $\mathbf{z} \sim \mathcal{N}(\mu, \sigma)$ of reduced dimensionality n_Z. This latent vector is concatenated with the 3D point $\mathbf{a}$ and processed by

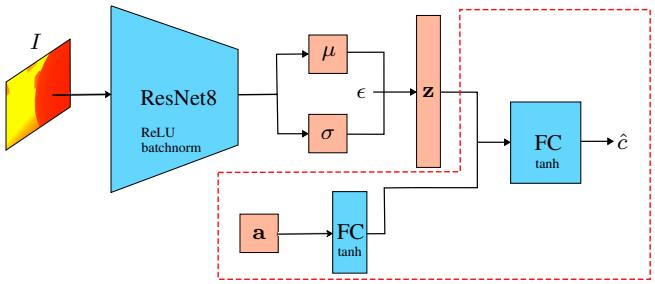

Fig. 1: Architecture of the classifier network. The image is encoded into a latent representation $\mathbf{z}$, which is sampled from a learned distribution $\mathbf{z} \sim \mathcal{N}(\mu, \sigma)$, using a randomness variable ϵ. The Fully Connected (FC) network is a coordinates-based NN, which outputs a predicted collision score $\hat{c}$ for input points $\mathbf{a} \in \mathbb{R}^3$ in the volumetric map described by I. The red lines highlight the part of the network to be embedded into the N-MPC.

a Fully Connected (FC) network which outputs the desired scalar approximate collision score $\hat{c}$.

Similar to [8], [14], we chose a ResNet-8 architecture for the CNN, using ReLU activations, batch normalizations and dropouts. The output volume of the last layer is passed through an average pooling layer, before flattening and dimension reduction with a pair of FC layer to compute the mean μ and standard deviation (std) σ of the latent encoding.

The FC network is trained as a coordinate-based FC network [21], [22], that takes as input a 3D point $\mathbf{a}$ and $\mathbf{z}$, and outputs $\hat{c}$. This network is chosen to be relatively shallow (3 hidden layers) to reduce gradient decay issues, and the layer widths are maintained small (≤ 128) for reducing the size of $\boldsymbol{\theta}$ and consequently the solving time of the N-MPC. It uses $\tanh$ activations, as the resulting FC network needs to be fully differentiable w.r.t. $\mathbf{a}$ to allow gradient-based optimization. A final sigmoid unit is employed to constrain its output in $[0,\ 1]$.

In order to reduce the dimensionality bias between $\mathbf{a}$ and $\mathbf{z}$ as input of the FC network (typically, 3 against 128), we process $\mathbf{a}$ with a first FC layer before the concatenation with the latent vector.

B. Training Methodology

The overall network is trained as a classifier, using a Binary Cross Entropy (BCE) loss $\mathcal{L}_{\text{BCE}}$ between the network output $\hat{c}$ and the actual label c, which is computed using an algorithmic implementation of Eq. (4). It is weighted such that collision samples are given more importance in the training, in order to reduce the false negative rate. This renders the resulting classifier more conservative but minimizes the likelihood of unpredicted collisions. A β-scaled Kullback-Leibler divergence metric $\mathcal{L}_{\text{KL}}$ [23] is used to enforce that $\mathbf{z}$ follows a proper normal distribution fitting the true posterior, as commonly done for variational encoders.

The training loss function $\mathcal{L} = \mathcal{L}_{\text{BCE}} + \mathcal{L}_{\text{KL}}$ is given by

$$\mathcal{L}_{\text{BCE}} = \frac{-1}{nb} \sum_{i=1}^{b} \sum_{j=1}^{n} \lambda_1 c_{ij} \log \hat{c}_{ij} + \lambda_0 (1 - c_{ij})(1 - \log \hat{c}_{ij}), \tag{7}$$

$$\mathcal{L}_{\text{KL}} = -\frac{\beta_{\text{norm}}}{2} \sum_{i=0}^{b} 1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2, \tag{8}$$

where b is the number of batch elements, n is the number of points sampled per image, subscripts i and j denotes quantities being computed from inputs I_j and $\mathbf{a}_j$, λ_0 and λ_1 are respectively weights for the 0 and 1 classes, and β_{norm} is a scaling factor computed from n_Z, n_H and n_V, according to [23].

The NN is trained entirely on simulated depth images, obtained using Aerial Gym [24]. The generated environments are sampled to be slightly cluttered, with 20 objects of medium sizes randomly placed (and oriented) in 5m^3, along with 2 long pillars, also randomly placed, and 4 walls. A random 6D pose of the camera is sampled within each simulated environment. Training dataset totalizes 250k images before augmentation (randomized flipping, shifting, and noising).

To account for the fact that the network infers the collision label for a single 3D point instead of the full AR volume, the training labels are computed on depth images processed such that obstacles are "inflated" by the radius of the drone [25], adding a constant unknown bias to be learned.

The input positions are sampled uniformly in spherical coordinates, in a volume that is larger than the FoV $\mathbb{F}$, to ensure control on the NN output at the boundary, as per Eq. (5). In [21], the authors showed that uniform sampling provides the best consistency in a similar 3D volume reconstruction task, as other sampling methods tends to introduce a bias in the model. We chose a high number of points sampled per image (e.g., 10^4), in order to statistically ensure that some sampled points fall into small objects during training. We note that the actual input $\mathbf{a}$ of the NN is scaled by d_{max}, α_H and α_V, such that

$$\mathbf{a} = \frac{1}{d_{\text{max}}} \left[\frac{x}{\tan \alpha_H},\ \frac{y}{\tan \alpha_V},\ z \right]^\top. \tag{9}$$

Since the collision classifier is to be included as a safety constraint in the N-MPC, it is mandatory that the current position of the drone is classified as free, i.e. that there always exists a safe solution for the AR. Moreover, for numerical stability when hovering, the positions close to the current one must also be classified as safe. Therefore, a ball of small radius (typically, a couple of centimeters) is defined around O_C within which all states are labeled as safe. Throughout training, points are sampled in this ball for each image, ensuring that the NN is well conditioned in this volume.

IV. N-MPC with Deep Collision Prediction

A. Neural N-MPC

In order to track position or velocity trajectories without colliding with surrounding obstacles, the collision prediction NN is integrated into the N-MPC scheme, as constraints on the AR position. The prediction step of the N-MPC is independent of the CNN part of the network, as the image input I is fixed for a given optimization loop.

Therefore, we must write $\hat{c}$ as a closed-form (parametric) function of the N-MPC state vector $\mathbf{x}$. The 3D input of the NN for prediction steps of the N-MPC is the position of the camera at a given time t, $O_C(t)$, expressed in the frame

$\mathcal{F}_{C_0} = \mathcal{F}_C(t_0)$ at which the depth image was captured. It is a function of $\mathbf{x}$ given by:

$$\begin{aligned} {}^{C_0}\mathbf{p}_C &= {}^{C_0}\mathbf{R}_W \, {}^{W}\mathbf{p}_C + {}^{C_0}\mathbf{p}_W \\ &= {}^{W}\mathbf{R}_{C_0}^{\top} ({}^{W}\mathbf{R}_B \, {}^{B}\mathbf{p}_C + {}^{W}\mathbf{p}_B - {}^{W}\mathbf{p}_{C_0}) \end{aligned} \tag{10}$$

where ${}^{B}\mathbf{p}_C$ is fixed and known, ${}^{W}\mathbf{p}_B$ and ${}^{W}\mathbf{R}_B$ are functions of $\mathbf{x}$, and ${}^{W}\mathbf{p}_{C_0}$ and ${}^{W}\mathbf{R}_{C_0}$ are parameters computed from the pose of the AR at the moment the depth image is captured.

Then, for a given depth image I captured at $t = t_0$, we denote $g_{\boldsymbol{\theta},\mathbf{z}}({}^{C_0}\mathbf{p}_C)$ the quantity to constrain, that is, the output of the FC part of the NN evaluated on the latent representation $\mathbf{z}$ of I, and the 3D position of the camera over the receding horizon, expressed in $\mathcal{F}_{C_0}$.

When evaluating the collision avoidance constraint, in order to make it convex and avoid vanishing gradients, the terminal sigmoid activation is replaced by an exponential unit, utilizing that

$$\forall x \in \mathbb{R}, \quad \text{sigmoid}(x) < 0.5 \;\Leftrightarrow\; \frac{e^x}{2} < 0.5. \tag{11}$$

We remark that even though position information is required for collision avoidance, contrary to partial-state-based navigation policies [14], the prediction is performed locally, w.r.t. $\mathcal{F}_{C_0}$ which moves with the AR at high frequency (typically, 60Hz), therefore alleviating the drift issues inherent to map-based collision avoidance.

B. NonLinear Programming

The collision-aware trajectory-tracking N-MPC objective is defined by the minimization of the weighted square norm of an output vector $\mathbf{y}(\mathbf{x})$ w.r.t. its reference $\mathbf{y}_r$, denoted $\|\mathbf{y}(\mathbf{x}) - \mathbf{y}_{r,k}\|^2_{\mathbf{W}}$. Such reference trajectory $\mathbf{y}_r$ is typically defined as sequences of position waypoints or velocity references, depending on the considered task. It is provided by a higher-level planner, e.g. based on a goal position to reach.

Additionally, the minimization of $g_{\boldsymbol{\theta},\mathbf{z}}({}^{C_0}\mathbf{p}_C)$ is included to the N-MPC cost function, in order to guide the N-MPC toward a solution that satisfies the obstacle avoidance constraint.

The discrete-time NLP over the receding horizon T, sampled in N shooting points, at a given instant t, given a depth image captured at $t_0 \leq t$ and compressed into a latent vector $\mathbf{z}$, is expressed as

$$\min_{\substack{\mathbf{x}_0 \dots \mathbf{x}_N \\ \mathbf{u}_0 \dots \mathbf{u}_{N-1}}} \quad \sum_{k=0}^{N} \|\mathbf{y}(\mathbf{x}_k) - \mathbf{y}_{r,k}\|^2_{\mathbf{W}} + w \; g_{\boldsymbol{\theta},\mathbf{z}}({}^{C_0}\mathbf{p}_{C,k}) \tag{12a}$$

$$\begin{aligned} s.t. \quad & \mathbf{x}_0 = \mathbf{x}(t) && && \text{(12b)} \\ & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), && k \in \{0, N-1\} && \text{(12c)} \\ & 0 \leq v_{x,k} \leq \overline{v_x}, && k \in \{0, N\} && \text{(12d)} \\ & \underline{\mathbf{u}} \leq \mathbf{u}_k \leq \overline{\mathbf{u}}, && k \in \{0, N-1\} && \text{(12e)} \\ & g_{\boldsymbol{\theta},\mathbf{z}}({}^{C_0}\mathbf{p}_{C,k}) \leq 0.5, && k \in \{0, N\} && \text{(12f)} \end{aligned}$$

where $\mathbf{x}(t)$ is the state estimate at t, $\mathbf{f}$ synthetically denotes the dynamics defined in Eq. (1), $\mathbf{W}$ and w are tunable weights, and $\underline{\bullet}$ and $\overline{\bullet}$ denotes mission-related lower and upper bounds on the N-MPC state and inputs.

Fig. 2: The AR used in the reported experiment, LMF, and a view of the (filled) depth image from its onboard front-facing camera.

	Accuracy		Precision		Recall	
	mean	*std*	*mean*	*std*	*mean*	*std*
Simulated images	93.1%	4.2%	86.0%	10.0%	98.6%	2.9%
Real images	95.4%	3.4%	92.1%	6.8%	98.3%	2.4%

TABLE I: Metrics of the NN collision classifier (mean and std).

V. Validation

A. Collision Classifier

This section presents a quantitative analysis of the collision classifier. A testing set of 5000 simulated images are gathered, within which 4M points are randomly sampled (≈ 0.2 points/cm^3). Accuracy, precision, and recall of the NN classifier are computed for each image and reported in Tab. I. The threshold for computing the metrics is 0.5, to fit the chosen value defined as the upper bound of the N-MPC constraint.

To assess the sim-to-real gap of the method, the NN is also evaluated on a set of real images captured with a Realsense d455 camera. We make use of evaluation images used in [26], i.e. a dataset of 1498 images captured in confined spaces, indoor rooms, long corridors, and outdoor environments with trees. The systematic errors in depth images (stereo shadow) are compensated with a filling algorithm [27].

The weighting of the BCE mentioned in Sec. III-B is, as expected, inducing a high recall of the classifier, to the detriment of precision. The metrics computed on real images are higher than for simulated data. This is explained by the fact that metrics are computed against the filled depth image, which is blurred during preprocessing. The resulting collision volume is thus smoothed and easier to approximate by the NN. Moreover, the real environments are inherently more structured than the chaotic synthetic randomly sampled simulation environments. The good performances on real images demonstrate the pertinence of the simulation-trained method and its applicability to real-world scenarios, as shown in Sec. V-D.

B. Setup

The proposed N-MPC is implemented in Python using Acados [28] and Casadi [29]. The neural network implementation is written with PyTorch, and it is interfaced with the N-MPC using ML-Casadi [20]. The NLP is transformed into a SQP solved with a Real-Time Iteration (RTI) scheme. The N-MPC inputs $\mathbf{u}$ is transformed into velocity commands and sent to the simulated or real system, e.g. through ROS or GenoM [30], which handles state estimation and low-level control. A goal waypoint is provided to the controller, which computes before each iteration a reference velocity vector of constant norm to go toward this point. Both in simulation

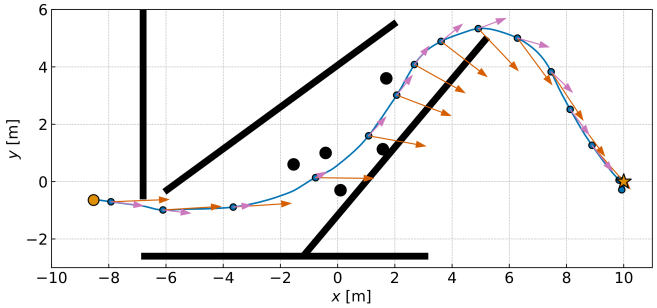

Fig. 3: (x, y) motion of the AR, in blue, among the black obstacles. The orange circle and stars are the initial and goal positions. The blue dots are the position of the AR every 2s, while the red and pink arrows are resp. the reference and actual velocity vector at the corresponding time.

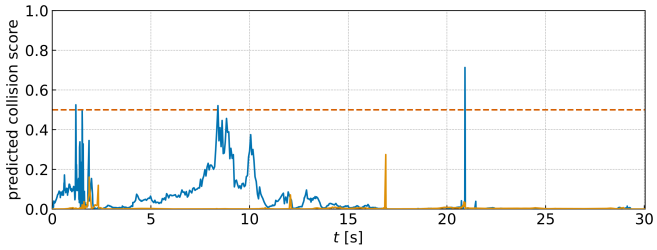

Fig. 4: Predicted collision score for the first (orange) and last (blue) shooting points of the receding horizon, throughout the trajectory, with its corresponding upper bound (red).

and experiments, the receding horizon is set to $T = 4$s, sampled in $N = 10$ points. The code for the NN training and inference, the N-MPC controller, and the ROS interface are released as open-source [1].

C. Gazebo Simulations

We first present some simulation result of the proposed framework. The AR is tasked to reach a waypoint behind a corridor filled with pillars. To showcase the avoidance behavior yield by the method, the N-MPC is weighted to maintain a constant z, while tracking a (x, y) velocity vector (of constant norm) toward the final waypoint. The resulting motion is reported in Fig. 3 and can be seen in the attached video. The predicted collision score throughout the trajectory is reported in Fig. 4. It displays that minor predicted violations of the constraint occur, which is a result of the input image noise creating sudden changes on the collision map. This makes the N-MPC solver warm-starting to be far from the optimal solution, preventing the RTI step from properly approximating it. Those violations are however immediately corrected in subsequent solving steps, preventing collision scores for the near future to increase.

In particular, we can observe at the end of the corridor that the actual velocity of the AR is orthogonal to the reference one, clearly demonstrating a non-greedy behavior. This illustrates the advantages of utilizing the N-MPC as a local planner, as its predicting capabilities allow to overcome local minima.

[1] https://github.com/ntnu-arl/colpred_nmpc

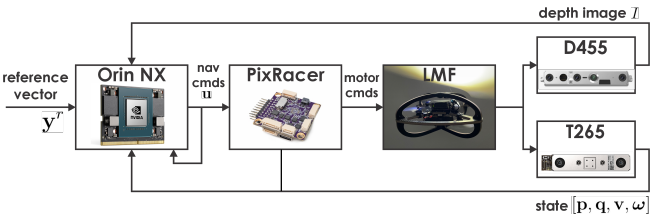

Fig. 5: Block diagram of the LMF aerial robot.

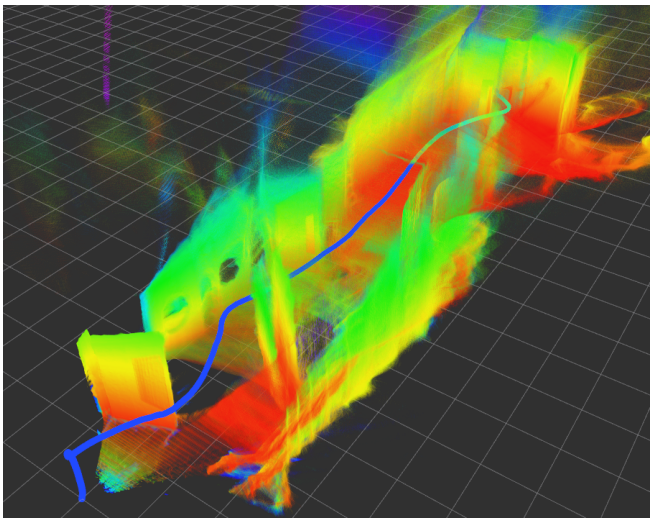

Fig. 6: Trajectory (blue line) of LMF avoiding obstacles. The goal position is located 15m in front of the starting point. The colored dots are the aggregated pointclouds captured by the D455 camera.

D. Experiments

1) System Overview: To evaluate the method, we utilize the Learning-based Micro Flyer (LMF) [14], [26], pictured in Fig. 2. The robot has a diameter of 0.43m and weights 1.2kg. It features a Realsense D455 for depth data at 480×270 pixel resolution and 30 FPS, a PixRacer Ardupilot-based autopilot for velocity and yaw-rate control, and a Realsense T265 fused with the autopilot's IMU for acquiring the robot's odometry state estimate. The robot integrates an NVIDIA Orin NX onboard in which the proposed method is executed by exploiting its GPU (for the CNN) and CPU (for the N-MPC). The system is depicted in Figure 5.

2) Experimental results: The experiment conducted with LMF consists of reaching a goal position located 15m in front of the starting location. The reference velocity is 1.5m/s. Several obstacles are present in the path of the AR, such that avoidance maneuvers are required. The final waypoint is chosen to be behind a wall. The resulting motion is summarized in Fig. 6 and can be seen in the attached video.

Table II presents the statistics of computation time onboard the Orin NX of our algorithms. Those data are gather throughout 15 missions in setups similar to the one showcased in Fig. 6, totalizing an aggregated flight time of 10 minutes. The core of the method (encoding of depth images and N-MPC solving) runs in less than 10ms, therefore reaching a maximum control frequency of 100Hz. The N-MPC solving time is consistently around 3ms, with 99% of samples below 4.3ms although there exist some outliers

Process	depth filling	CNN	N-MPC
Real images	11.85	6.85	3.09

TABLE II: Average computation times (in ms) of the main parts of the method on the onboard computer.

(with a recorded maximum of 10.3ms). Despite the depth filling algorithm [27] being rather slow because working on CPU, the overall method remains real time and faster than the odometry feedback of the autopilot.

VI. Conclusion

In this work, we proposed a new paradigm for collision avoidance that merges NN-based image data processing with N-MPC. The NN enables the framework to process in real time depth images that are used by the optimal controller in a cascaded fashion. This network is divided in two parts. First, a convolution variational encoder compresses the input image into a latent vector that encompasses collision information. The second part is a FC network which is written as an algebraic function of the N-MPC state, constraining the predicted position to avoid perceived obstacles. The proposed method is first quantitatively evaluated on its classification performances, with emphasis on assessing the performance of the simulation-trained network on real images. Then the resulting N-MPC controller is evaluated on simulation and experiments, showcasing avoidance behaviors and high frequency on the onboard computer, despite utilizing the neural networks. The one-by-one image processing is however constraining the motion to belong to the current FoV, preventing exploitation of the full dynamics of the AR. Thus, future works include exploring memory-based neural network to encode information from several short-past images, enabling the N-MPC to plan in a less constrained space. The injection of real data in the training could also be explored to render the NN robust to stereo errors, such as shadowing, and thus discard the need for a time-consuming and error-prone depth-filling algorithm. Finally, future works should also provide in-depth performance comparison with other existing sensor-based collision avoidance methods, as well as map-based path planners.

References

[1] Y. Tian, K. Liu, K. Ok, L. Tran, D. Allen, N. Roy, and J. P. How, "Search and rescue under the forest canopy using multiple UAVs," *The Int. Journal of Robotics Research*, vol. 39, no. 10-11, pp. 1201–1221, 2020.

[2] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph-based subterranean exploration path planning using aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020.

[3] P. Petracek, V. Kratky, and M. Saska, "Dronument: System for reliable deployment of micro aerial vehicles in dark areas of large historical monuments," *IEEE Robotics and Automation Letters*, vol. 5, pp. 2078–2085, 2020.

[4] P. Zhang, Y. Zhong, and X. Li, "SlimYOLOv3: Narrower, faster and better for real-time UAV applications," in *2020 IEEE/CVF Int. Conf. on Computer Vision*, 2019.

[5] Y. Akbari, N. Almaadeed, S. Al-maadeed, and O. Elharrouss, "Applications, databases and open computer vision research from drone videos and images: a survey," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3887–3938, 2021.

[6] S. Khattak, H. Nguyen, F. Mascarich, T. Dang, and K. Alexis, "Complementary multi–modal sensor fusion for resilient robot pose estimation in subterranean environments," in *2020 Int. Conf. on Unmanned Aircraft Systems*, 2020, pp. 1024–1029.

[7] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation." in *2017 IEEE Int. Conf. on Robotics and Automation*, 2017, pp. 5759–5765.

[8] 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, pp. 1088–1095, 2018.

[9] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, 2021.

[10] V. Tolani, S. Bansal, A. Faust, and C. Tomlin, "Visual navigation among humans with optimal control as a supervisor," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2288–2295, 2021.

[11] G. Kahn, P. Abbeel, and S. Levine, "BADGR: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[12] H. I. Ugurlu, X. H. Pham, and E. Kayacan, "Sim-to-real deep reinforcement learning for safe end-to-end planning of aerial robots," *Robotics*, vol. 11, no. 5, p. 109, 2022.

[13] D. Hoeller, L. Wellhausen, F. Farshidian, and M. Hutter, "Learning a state representation and navigation in cluttered and dynamic environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5081–5088, 2021.

[14] H. Nguyen, S. H. Fyhn, P. De Petris, and K. Alexis, "Motion primitives-based navigation planning using deep collision prediction," in *2022 IEEE Int. Conf. on Robotics and Automation*, 2022, pp. 9660–9667.

[15] 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. 7976, pp. 982–987, 2023.

[16] D. C. Tan, F. Acero, R. McCarthy, D. Kanoulas, and Z. A. Li, "Value functions are control barrier functions: Verification of safe policies using control theory," in *2nd Workshop on Formal Verification and Machine Learning*, 2023. [Online]. Available: https://arxiv.org/abs/2306.04026

[17] C. Dawson, B. Lowenkamp, D. Goff, and C. Fan, "Learning safe, generalizable perception-based hybrid control with certificates," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1904–1911, 2022.

[18] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," 2017, pp. 1714–1721.

[19] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh, "KNODE-MPC: A knowledge-based data-driven predictive control framework for aerial robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2819–2826, 2022.

[20] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural MPC: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2397–2404, 2023.

[21] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *2019 IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 4460–4470.

[22] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," pp. 7537–7547, 2020.

[23] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-VAE: Learning basic visual concepts with a constrained variational framework," in *2017 Int. Conf. on Learning Representations*, 2017.

[24] M. Kulkarni, T. J. Forgaard, and K. Alexis, "Aerial gym–isaac gym simulator for aerial robots," in *"The Role of Robotics Simulators for Unmanned Aerial Vehicles" Workshop at 2023 IEEE Int. Conf. on Robotics and Automation*, 2023. [Online]. Available: https://arxiv.org/abs/2305.16510

[25] M. Kulkarni and K. Alexis, "Task-driven compression for collision encoding based on depth images," in *2023 Int. Symp. on Visual Computing*, 2023.

[26] M. Kulkarni, H. Nguyen, and K. Alexis, "Semantically-enhanced deep collision prediction for autonomous navigation using aerial robots," in *2023 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2023.

[27] J. Ku, A. Harakeh, and S. L. Waslander, "In defense of classical image

processing: Fast depth completion on the CPU," in *2018 15th Conf. on Computer and Robot Vision*, 2018.

[28] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, vol. 14, p. 147–183, 2021.

[29] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[30] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, and F. Ingrand, "GenoM3: Building middleware-independent robotic components," in *2010 IEEE Int. Conf. on Robotics and Automation*, 2010, pp. 4627–4632.