Motion Primitives-based Navigation Planning using Deep Collision Prediction

Huan Nguyen, Sondre Holm Fyhn, Paolo De Petris, and Kostas Alexis

Abstract—This paper contributes a method to design a novel navigation planner exploiting a learning-based collision prediction network. The neural network is tasked to predict the collision cost of each action sequence in a predefined motion primitives library in the robot’s velocity-steering angle space, given only the current depth image and the estimated linear and angular velocities of the robot. Furthermore, we account for the uncertainty of the robot’s partial state by utilizing the Unscented Transform and the uncertainty of the neural network model by using Monte Carlo dropout. The uncertainty-aware collision cost is then combined with the goal direction given by a global planner in order to determine the best action sequence to execute in a receding horizon manner. To demonstrate the method, we develop a resilient small flying robot integrating lightweight sensing and computing resources. A set of simulation and experimental studies, including a field deployment, in both cluttered and perceptually-challenging environments is conducted to evaluate the quality of the prediction network and the performance of the proposed planner.

I. INTRODUCTION

Recent advances in the field of aerial robotics have enabled their utilization in various applications including industrial inspection [1, 2], search and rescue [3], surveillance [4], subterranean exploration [5], and more. Despite the progress, the task of autonomous navigation in complex, perceptually-degraded environments, such as obstacle-filled industrial settings, and underground mines remains particularly challenging. In such settings, the underlying robot localization and mapping may be subject to significant uncertainty and drift [6–8], and requires high computational cost [9], while consistent high-resolution mapping can be particularly demanding. In this context, ensuring collision-free navigation can be a gruelling task and remains an open problem.

In this work, we aim to take a different approach to allow safe navigation of aerial robots by departing from the classical techniques that require a continuously maintained and updated map of the environment within which the robot localizes [10]. Employing a data-driven approach to allow collision-free navigation without explicit map building and an accurate estimate of the robot’s position, we propose a local mOtions pRimitives-bAsed navigation planner using a deep CoLLision prEdictor (ORACLE) that further accounts for the goal direction provided by a global planner. The collision predictor is a deep neural network taking as inputs the current depth image, estimated linear and angular velocities and associated covariance, as well as action sequences from a fixed motion primitives library based on forward velocity and steering angle commands. It evaluates - in parallel - the collision probability for each action sequence, based on which the safest motion primitives are chosen and the corresponding goal-reaching costs are calculated to determine the best action in a receding horizon fashion.

The proposed approach was experimentally evaluated in two unseen real-life environments, while the complete method was designed and trained purely in simulation. As a testbed for our learning-based navigation policy, we developed a new collision-tolerant aerial robot, called Learning-based Micro Flyer (LMF). LMF integrates a lightweight, and low-cost sensor suite including an Intel Realsense D455 RGB-D sensor, an Intel Realsense T265 visual-inertial module, and a 1D range sensor, as well as an autopilot, and an NVIDIA Xavier NX CPU/GPU board. The first experiment was conducted within obstacle-filled building corridors and aimed to verify the ability of the method to avoid novel obstacles, while navigating using goal directions commanded by a global planner. The second experiment was carried out in an underground mine environment near Trondheim, Norway, demonstrating the ability of the robot to safely navigate in a narrow, relatively dark environment that can lead to high uncertainty in visual-inertial estimation.

The remainder of this paper is organized as follows: Section II presents related work, followed by the problem statement in Section III. The proposed learning-based navigation policy is presented in Section IV. Evaluation studies are detailed in Section V followed by conclusions in Section VI.
II. RELATED WORK

A set of contributions in a) planning with neural networks, b) planning under uncertainty and modeling uncertainty in deep neural networks, and c) planning with motion primitives relate to this work.

A. Planning with Neural Networks

A set of works have applied deep learning to the problem of autonomous robot navigation. Specifically, the problem of navigation with RGB and depth cameras has attracted increased attention as these sensors are low-cost, low-power and lightweight. The authors in [11, 12] use supervised learning, while the work in [13] utilizes reinforcement learning and domain randomization with RGB image inputs to learn reactive navigation policies. While the robot’s state is often available for the low-level control, these works do not utilize the current estimated state of the robot to make predictions although this can lead to better performance. On the other hand, the authors in [14] use the current robot’s linear and angular velocities in addition to the latest camera frame to generate position setpoints. However, position feedback is not always available or reliable, especially in perceptually degraded environments [6].

B. Planning under Uncertainty and Modeling Uncertainty in Deep Learning

Modeling uncertainty is essential to achieve safe planning in environments where the state of the robot is highly uncertain [19]. Moreover, when using deep neural networks for making predictions, there are two kinds of uncertainty that need to be considered: aleatoric uncertainty which captures noise inherent in the observations, and epistemic uncertainty which accounts for model uncertainty [20]. Most existing works applying deep neural networks for autonomous navigation account for epistemic uncertainty only, for instance by using autoencoders [21], dropout and bootstrap [22, 23]. One of the exceptions is [24] which accounts for both uncertainty in the sensor observations and model uncertainty. However, the method for propagating aleatoric uncertainty through sigmoid and tanh activation layers, which are usually used in recurrent neural networks, is not presented. In this work, we approximately propagate the robot’s state uncertainty through the prediction network using the Unscented Transform (UT) [25] as in [26], and use Monte Carlo (MC) dropout [27, 28] to account for the model uncertainty.

C. Planning with Motion Primitives

A motion primitives library [29] can be generated by either sampling the vehicle’s configuration space or its control space [30]. Depending on what type of low-level controller is implemented, control space primitives for multirotors can be generated by sampling jerk [31], acceleration [32, 33], or other input signals. The authors in [34] use configuration space primitives proposed in [35] to generate minimum-jerk primitives online, however, position feedback is required and not all generated trajectories are guaranteed to lie within the sensor’s FOV. Our work follows approaches like those proposed in [36, 37] to sample the desired speed and steering angle, where possible steering angles are uniformly sampled from within the FOV.

III. PROBLEM FORMULATION

The broader problem considered in this work is that of autonomous collision-free aerial robot navigation assuming no access to the maps of the environment (from offline data or online reconstruction) and no information for the robot position but only a partial state estimate of the robot’s orientation and linear/angular velocities combined with real-time depth data within an angle- and range-constrained sensor frustum. We assume that there is a global planner providing goal directions to the robot (e.g., for exploration), possibly by having access to a topological map of the environment, and focus on designing a local safe navigation planner to avoid obstacles while following the goal directions. Let $\mathcal{B}$ be the body frame of the robot, $o_t$ be the current depth image, $s_t = [v_t, \omega_t]^T$ the estimated partial state of the robot including the forward velocity in $\mathcal{B}$ ($v_t$) and the angular velocity around $z$-axis of $\mathcal{B}$ ($\omega_t$), $\Sigma_t = \text{diag}([\sigma^2_{v,t}, \sigma^2_{\omega,t}])$ the covariance of the estimated robot’s partial state, $\psi^g_t$ the goal heading direction given by the global planner, and $a_{t:t+H} = [a_t, a_{t+1}, \ldots, a_{t+H-1}]$ an action sequence having length $H$ where the action at time step $t + i$ ($i = 0, \ldots, H - 1$) includes the reference forward speed $v^r_{t+i}$ and the steering angle from the current yaw angle of the robot $\psi^r_{t+i} = (a_{t+i}, \psi^g_{t+i})^T$. The exact problem considered is then formulated as that of finding an optimized collision-free sequence of actions $a_{t:t+H}$ enabling the robot to safely move towards the goal direction $\psi^g_t$ given $(o_t, s_t, \Sigma_t)$.

IV. PROPOSED APPROACH

The proposed mOnetion PrImitives-bAsed navigation planner using a deep CoLLision prEdictor (ORACLE) is detailed in this section. Assuming only access to estimates of the robot’s linear and angular velocities, alongside the depth image, ORACLE identifies the next best sequence of actions (velocity-steering commands) that ensure that the system is navigating towards a goal heading direction, while avoiding the obstacles in its environment. The first action of this sequence is executed by the robot, while the process continues iteratively. Importantly, the “global” goal direction may be provided by any global planner thus allowing ORACLE to be combined with any high level planning framework. Figure 2 provides an overview of the method’s architecture.
A. Velocity-Steering Angle Motion Primitives Library

In this work, each candidate action sequence is a time sequence of velocity and steering commands derived from a predefined Motion Primitives Library (MPL) and has fixed forward speed and fixed steering angle sampled within the FOV of the depth sensor for every time step $v_{t+i} = v_r$, $\psi_{t+i} = \psi_r$ ($i = 0, \ldots, H - 1$). Specifically, we denote $a_{t+i}^k$ as the $k$th action sequence in the MPL. Each planned sequence is such that it guarantees that the trajectories always lie in the FOV of the sensor. As a notable distinction to other MPL-based methods, the planned sequences do not include the robot’s position space but remain in velocity/steering angle space as the underlying assumption is that ORACLE does not have access to a position estimate.

B. Learning-based Collision Prediction Network (CPN)

At the core of ORACLE is a neural network that processes a) the input depth image $o_t$, b) the robot’s partial state $s_t$, and c) motion primitives-based sequences of future velocity and steering angle references $a_{t+i}$, and is trained to predict the collision probabilities of the anticipated robot motion at each time step from $t+1$ to $t+H$ in the future $\hat{c}_{col} = \{\hat{c}_{col+1}, \hat{c}_{col+2}, \ldots, \hat{c}_{col+H}\}$ by entirely using collision data in simulation. Thus, ORACLE avoids the need for hand-engineered collision checking algorithms such as [34, 38] or access to a reconstructed map of the environment [39, 40].

Since real-life depth images are often subject to several shortcomings compared to simulated data, including a) missing information, b) loss of detail, and c) depth noise [18], we perform an additional filtering step using the IP-Basic algorithm [41] to refine the depth frame and thus reduce the mismatch between the real and simulated depth images. The collision costs for every action sequence in the motion primitives library of velocity-steering commands can then be evaluated in parallel as per [42] exploiting modern GPU architectures and thus enabling high update rate compute. Notably, when evaluating the collision costs, ORACLE does not only consider the mean estimate of the robot’s partial state but also the estimated uncertainty, as well as the uncertainty in the neural network model.

In further detail, the filtered depth image $o_f^t$ of size $270 \times 480$ is processed by a Convolutional Neural Network (CNN) branch, while the estimated partial state of the robot is processed by a Fully-Connected (FC) network before concatenating it with the output feature vector from the CNN branch. The concatenated vector is then fed to another FC network before being used as the initial hidden state of a Long Short-Term Memory (LSTM) network. The input to the LSTM cells is generated by the velocity-steering angle action sequence provided by the motion primitives library, while the outputs of the LSTM cells are passed through a FC network to predict the collision label at each time step. The prediction network architecture, shown in Figure 3, is inspired by the network in [16]. However, we replace the MobileNetV2 part with the ResNet8 network with dropout layers as in [12] for faster inference speed and model uncertainty estimation using MC dropout as in [28]. For the type of training considered, typically there are more negative (non-collision) labels than positive (collision) labels in the dataset. Hence, we use a higher weight for positive labels to ensure high recall which leads to a safer behaviour. The network is trained end-to-end with weighted binary cross-entropy loss using the Adam optimizer.

C. Uncertainty-aware Prediction

As mentioned, ORACLE further considers the uncertainty of the robot’s partial state and the model uncertainty of the collision prediction network. First, we calculate the final collision cost for each action sequence in the motion primitives library as the weighted sum of the collision probabilities at each time step. Specifically, the sooner the collision event is
predicted to happen, the more it will contribute to the final collision cost:

\[ \hat{c}_{\text{col}} = \sum_{i=1}^{H} \hat{c}_{i+1}^{\text{col}} e^{-\lambda(i-1)}, \quad \lambda > 0 \]  

(1)

To account for the uncertainty of \( s_t \), which cannot be negligible - especially in fast flight or within perceptually degraded environments - we utilize the Unscented Transform (UT) [25] to approximately propagate the uncertainty in \( s_t \) to the predicted collision cost \( \hat{c}_{\text{col}} \) of an action sequence \( a_{t:t+H} \). In the UT, for a \( k \)-dimensional robot state, \( \Sigma_{s} = 2k + 1 \) sigma points, and their associated weights, are computed based on the mean value \( s_t \) and covariance \( \Sigma_{s} \). These sigma points are then propagated through the CPN, and the mean and variance of the output distribution of \( \hat{c}_{\text{col}} \) are calculated based on the output predictions of the sigma points.

Additionally, the model uncertainty - which can be significant in novel environments - can be captured by performing MC dropout for the network as in [28]. For efficient neural network forward pass and uncertainty estimation, we only perform MC dropout in the CNN branch and split the neural network shown in Figure 3 into 3 parts, namely the CNN, Combiner, and Prediction networks. Let a) \( N_{MC}, N_{MP} \) be the number of dropout masks and action sequences in the MPL, respectively, and b) \( \sigma_{n}^{\text{col}}, \mu_{n}^{\text{col}} (n = 1, \ldots, N_{MC}) \) the variance and mean of the predicted collision cost of \( a_{t:t+H} \), estimated by the UT with different dropout masks of the CNN part. The total variance can then be expressed as:

\[ \sigma_{\text{tot}}^{\text{col}} = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} \left[ \sigma_{n}^{\text{col}} + (\mu_{n}^{\text{col}} - \bar{\mu}_{\text{col}})^2 \right] \]  

(2)

where \( \bar{\mu}_{\text{col}} = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} \mu_{n}^{\text{col}} \). The final uncertainty-aware collision cost for an action sequence is given as:

\[ \hat{c}_{\text{uac}} = \bar{\mu}_{\text{col}} + \alpha \sqrt{\sigma_{\text{tot}}^{\text{col}}}, \quad \alpha > 0 \]  

(3)

Specifically, we denote \( \hat{c}_{\text{uac}} \) as the uncertainty-aware collision cost of the \( k \)-th action sequence \( a_{t:t+H}^{k} \) in the MPL \( (k = 1, \ldots, N_{MP}) \). By splitting the prediction network into 3 parts, we can perform inference on the ensemble of CNNs, the Combiner and Prediction networks with different batch sizes of 1, \( N_{MC} \times N_{\Sigma} \), and \( N_{MC} \times N_{\Sigma} \times N_{MP} \), respectively, avoiding the need to run the forward passes through the whole CPN \( N_{MC} \times N_{\Sigma} \times N_{MP} \) times.

D. Learning-based Navigation Policy

After calculating the uncertainty-aware predicted collision cost for each action sequence in the motion primitives library, as described in section IV.C, the minimum collision cost \( c_{\text{min}} \) of all action sequences is calculated, and all the action sequences having collision cost greater than \( \hat{c}_{\text{uac}} = c_{\text{min}} + c_{th} \), where \( c_{th} \) is a set positive threshold, are discarded. The remaining action sequences are checked for deviation from the global planner’s goal direction \( \psi_{t} \):

\[ c_{\text{goal}} = |\text{wrap}(\psi_{t}^r + \psi_t - \psi_{t}^g)| \]  

(4)

where \( \psi_{t}^r \) is the current yaw angle of the robot and \( \text{wrap}(\cdot) \) is the function that wraps an angle in radians to \([ -\pi, \pi] \). The best action sequence is chosen and its first step is executed, while the whole procedure is repeated in a receding horizon fashion. Algorithm 1 outlines ORACLE’s key steps.

Algorithm 1 ORACLE Navigation Planner

1: \( s_t, \Sigma_{s} \leftarrow \) Get robot’s partial state and covariance
2: \( m_i \) (\( i = 1, \ldots, N_{\Sigma} \)) \( \leftarrow \) sigma points given by UT
3: \( o_t^* \leftarrow \) Get filtered depth image
4: \( \text{for} \ n = 1 \text{ to } N_{MC} \text{ do} \)
5: \( \text{CPN}_n \leftarrow \) Apply dropout mask \( n \) to CPN
6: \( \text{for} \ a_{t:t+H} \text{ in MPL do} \)
7: \( \text{for} \ i = 1 \text{ to } N_{\Sigma} \text{ do} \)
8: \( \text{// Parallel evaluation in GPU} \)
9: \( \hat{c}_{i+1:t+H+1}^{\text{col}} \leftarrow \text{CPN}_n(o_t^*, m_i, a_{t:t+H}) \)
10: \( \text{Calculate } c_{\text{col}} \text{ by Eq. (1)} \)
11: \( \text{end for} \)
12: \( \text{Calculate } c_{n}^{\text{col}}, \mu_{n}^{\text{col}} \text{ by UT on } \{ c_{\text{col}} \}_{N_{\Sigma} \times 1} \)
13: \( \text{end for} \)
14: \( \text{end for} \)
15: \( \text{for} \ a_{t:t+H} \text{ in MPL do} \)
16: \( \text{Calculate } c_{\text{uac}} \text{ by Eq. (2), (3)} \)
17: \( \text{for} \ \hat{c}_{\text{uac}} \leftarrow \min_{k=1,\ldots,N_{MC}} \{ \hat{c}_{\text{uac}} \} \)
19: \( \text{for} \ a_{t:t+H} \text{ in MPL having } c_{\text{uac}} < \hat{c}_{\text{min}} + c_{th} \text{ do} \)
20: \( \text{Calculate } c_{\text{goal}} \text{ by Eq. (4)} \)
21: \( \text{end for} \)
22: \( \text{Choose } a_{t}^{*} \text{ in MPL having the smallest } c_{\text{goal}} \)
23: \( \text{Execute the first action } a_{t}^{*} \text{ then go to step 1} \)

E. Implementation Details and Data Collection

The RotorS simulator [43] is used to collect data for training the collision predictor network. To ensure successful sim-to-real transfer, the dynamics of the low-level attitude and velocity controllers of the quadrotor model used in the simulator are tuned to match the real system. Relevant methods for dynamic system identification of MAVs are presented in [44]. To collect data for predicting collision probabilities at the future time steps, an action sequence with random \( v^r \) and \( \psi^r \) within the sensor’s FOV is drawn and is fully executed. This process is repeated until the robot collides with the obstacles or a timeout event occurs. One training data point \( d \) is recorded every time the robot moves more than \( \Delta_{th} \) meters, having the format \( d = (o_t, s_t, a_{t:t+H}, c_{\text{col}}^{t+1:t+H+1}) \) where \( c_{\text{col}}^{t+1:t+H+1} = [c_{t+1}^{\text{col}}, c_{t+2}^{\text{col}}, \ldots, c_{t+H}^{\text{col}}] \). \( c_{\text{col}}^{t+1} \) denotes the collision label at time step \( t+1, i = 1, \ldots, H \) (equal to 1 for collision and 0 for non-collision status). When the collision happens midway an action sequence, for instance after the execution of \( a_{t+k} \) \( (k \leq H) \), then the collision labels of the remaining time steps \( c_{\text{col}}^{t+k+1:t+H} \) are set to 1, and augmented data points are also added to the dataset by replacing the actions after \( a_{t+k} \) with randomly sampled actions as in [16]. The number of data points created by augmenting the remaining actions is such that the number of data points with no collision label and the number of data points with at least...
one collision label are almost equal. Moreover, given that
the robot dynamics are holonomic and effectively invariant
to its heading orientation, we also perform horizontal flip
data augmentation, specifically by adding the augmented data
point

\[ d_{\text{flip}}(o_t, s_t, \omega_t) = (o_t^{\text{flip}}, s_t^{\text{flip}}, \omega_t^{\text{flip}}, \omega_{t+1:2t+H}^{\text{flip}}, \omega_{t+2H+1}^{\text{flip}}) \]

where \( o_t^{\text{flip}} \) is the horizontally flipped image of \( o_t \), \( s_t^{\text{flip}} \) and \( \omega_t^{\text{flip}} \)
are created by changing the sign of \( \omega_t \) and \( \psi_t^{\text{flip}} \) (\( i = 0, \ldots, H - 1 \) ) in \( s_t \) and \( a_{t+1:2t+H} \), respectively.

In order to collect a comprehensive dataset for training the
collision predictor, we randomized the initial position and
orientation of the robot, as well as the obstacles’ locations,
categories, dimensions, and densities in order to collect
around 1.5 million data points in total. Figure 3 illustrates
one indicative training environment. The derived dataset was
then split into a training and validation subset. Notably, a
particular limitation of the depth sensor used in real-life was
taken into account. Specifically, to tackle the problem that
depth cameras often present gaps in depth data when facing a
reflective surface [45], we designed artificial obstacles having
holes with sizes that could “just” fit our flying robot, or
smaller, to enhance the trained network’s tendency to avoid
holes with sizes that could “just” fit our flying robot, or
reflective surface [45], we designed artificial obstacles having
taken into account. Specifically, to tackle the problem that
particular limitation of the depth sensor used in real-life was
one indicative training environment. The derived dataset was
around 1.5 million data points in total, Figure 4.i illustrates
orientation of the robot, as well as the obstacles’ locations,
collision predictor, we randomized the initial position and
the proposed learning-based navigation planner.

A. Simulation Studies

To evaluate ORACLE’s ability to negotiate novel environ-
ments in combination with highly degraded state estimates,
we conducted simulation studies and compare it with a more
“naive” version which utilizes the CPN directly to calculate
the collision cost without considering the uncertainty of
or that of the neural network model as in [15]. Accordingly,
the naive planner is not using the UT samples and the ensemble
of NNs (cf. Figure 2). Two environments are created to
verify the method’s performance, namely: a) environment A
(Figure 4ii) that contains obstacles seen during the training
phase and the robot is given perfect state estimation, while
b) environment B (Figure 4iii) contains unseen obstacles
and the robot is given largely deteriorated state estimation
\( s_{\text{deter}} = [v_{\text{deter}}, \omega_{\text{deter}}] \) (specifically \( v_{\text{deter}} = v_i - 1.0 \text{m/s}, \omega_{\text{deter}} = \omega_i + 0.1 \text{rad/s} \)). Both ORACLE and the naive planner are
deployed in each environment 20 times with different initial
conditions of the robot, a timeout period of 100 seconds,
and reference forward velocity 1.25m/s. The total number of
 collisions, and the average flight time before a collision are
reported in Table I. Both planners perform well in environ-
ment A with the average flight times almost equal to the
timeout period. However, the uncertainty-aware ORACLE
significantly outperforms the naive planner in environment
B where uncertainty-aware collision prediction is essential.

| Environment | Planner               | Collisions | Average flight time |
|-------------|-----------------------|------------|---------------------|
| A: Seen obstacles + perfect state | Naive | 0/20 | 100s |
| B: Novel obstacles + deteriorated state | Naive | 20/20 | 25.9s |

TABLE I

B. Experimental Studies

Two real-life experiments were conducted using the LMF
platform to evaluate the performance of ORACLE.

1) System Overview: LMF inherited the collision-tolerant
design of the Resilient Micro Flyer [46], yet with an
increased diameter at 0.43m and a mass of 1.2kg. It integrates
a RealSense D455 to obtain depth data, a nadir-facing TFMi-
1D range sensor, a PixRacer PX4-based autopilot delivering
attitude and thrust control, and a Realsense T265 fused with
the IMU of the autopilot allowing it to estimate the speed,
orientation and angular rate of the robot. The decision not
to assume access to a position estimate reflects our under-
standing of the challenges aerial robots encounter during fast
flight or within perceptually-degraded environments [6, 47].
The provided velocity estimates are then used in a fixed-
gain velocity controller. Similarly, two Proportional-Integral-
Derivative controllers for yaw control and height regulation
from the ground using the TFMi sensor are also developed.
The ORACLE planner and the low-level controllers are
implemented on a Jetson Xavier NX onboard LMF.

2) Experimental Results: In the first experiment, illus-
trated in Figure 5 LMF is tasked to follow the goal directions
given by a global planner, while navigating safely in a
corridor filled with various types of obstacles that were not
seen during training. Figure 5 presenta presents predictions of the
CPN at some specific scenarios, where the trajectories are
generated only for visualization purposes based on \( s_t \) and
the MPL using the estimated dynamics models of the velocity

Fig. 5. Main hardware components onboard LMF.

Fig. 4. i) Indicative simulation environments for collecting training data, ii,iii) environments for simulation studies evaluating ORACLE.
Fig. 6. Experiment in corridors filled with obstacles. The 4 plots on the left depict specific planning instances where the visualized trajectories are generated based on \( s_t \) and the MPL, the green trajectories correspond to the action sequences that pass the collision cost threshold check, while the blue trajectory corresponds to the best action sequence. The right figure presents the reconstructed map of the environment, which is not given to the robot, as well as the goal direction on each segment given by the global planner.

Fig. 7. Field experiment in an underground mine. The top figure illustrates the reconstructed map of the environment, which is not given to the robot. The two bottom rows depict the raw depth image \( o_t \) (a), the filtered image \( o'_t \) (b), and the visualized trajectories based on \( s_t \) and the MPL (c).

and yaw controllers mentioned in subsection IV-E. The green trajectories correspond to the action sequences that pass the collision cost threshold check, while the blue trajectories correspond to the best action sequence. As shown, the visualized trajectories correlate well with the collision cost predicted by the CPN, showing reliable performance of the CPN in various situations.

In the second experiment, LMF traveled a distance of 110m in a narrow underground environment, which is particularly challenging since the depth image data is not perfect, and the uncertainty in the estimated partial state of the robot is not negligible due to the fairly dark conditions which lead to relatively weak features. Figure 7.1 demonstrates the effect of the filtering process presented in [41], which reduces the sim-to-real gap and also removes false obstacles. Furthermore, Figure 7.2 demonstrates the benefit of including in the training environment obstacles with narrow holes that cannot fit the robot, which prevents LMF from erroneously steering to the fence on the right in this situation. At any time, the goal direction is commanded to be the current heading of the robot which in turn allows the system to pick the safe direction that deviates the least from the current heading direction of the robot in this unknown environment.

C. Computational Analysis

As outlined in Algorithm 1, most of the complexity terms of the ORACLE planner come from two operations, namely a) the image filtering process (Line 3) and b) multiple forward passes through the CPN (Lines 9). The inference time of the CPN includes the inference times of the ensemble of CNNs, the Combiner and Prediction networks. Assuming \( N_{MC} \) forward passes through the ensemble of CNNs, and \( N_{MC} \times N_{\Sigma} \) forward passes through the Combiner network can be run in parallel on the GPU, their complexities are \( O(1) \). On the other hand, the inference time of the Prediction network is dominated by the inference time of the LSTM network and has the complexity of \( O(H) \) [48], assuming that \( N_{MC} \times N_{\Sigma} \times N_{MP} \) forward passes through it can be run in parallel on the GPU. With \( N_{MC} = 5, N_{\Sigma} = 5, N_{MP} = 64, H = 18 \), on average, the ORACLE loop takes 47.4ms in which the inference time through the ensemble of CNNs, the Combiner and Predictor networks, and the filtering time is 6.2ms, 2.5ms, 8.4ms, and 20.1ms, respectively.

VI. CONCLUSIONS

In this work, a navigation planner based on a learning-based collision prediction network was proposed and experimentally verified using a custom-designed resilient micro flyer. The algorithm uses a neural network to predict the collision cost for each action sequence in a motion primitives library, and accounts for the uncertainty of the robot’s partial state and the neural network model. After thresholding the collision cost, a goal direction given by a global planner is used to choose the best action sequence to execute. A set of simulation and experimental studies is presented and serves to verify the performance of the proposed approach in fairly cluttered, narrow and visually challenging environments.
[39] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An efficient probabilistic 3D mapping framework based on octrees,” *Autonomous Robots*, 2013.

[40] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, “Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[41] J. Ku, A. Harakeh, and S. L. Waslander, “In defense of classical image processing: Fast depth completion on the cpu,” *2018 15th Conference on Computer and Robot Vision (CRV)*, pp. 16–22, 2018.

[42] J. C. Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust, “Neural collision clearance estimator for fast robot motion planning.” *CoRR*, vol. abs/1910.05917, 2019. [Online]. Available: [http://arxiv.org/abs/1910.05917](http://arxiv.org/abs/1910.05917)

[43] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, “Rotors-a modular gazebo mav simulator framework,” in *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[44] I. Sa, M. Kamel, R. Khanna, M. Popovic, J. Nieto, and R. Siegwart, “Dynamic system identification, and control for a cost effective open-source vtol mav,” in *11th Conference on Field and Service Robotics (FSR 2017)*, 2017.

[45] M. Popovic, F. Thomas, S. Papatheodorou, N. Funk, T. Vidal-Calleja, and S. Leutenegger, “Volumetric occupancy mapping with probabilistic depth completion for robotic navigation,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5072–5079, 2021.

[46] P. De Petris, H. Nguyen, T. Dang, F. Mascarich, and K. Alexis, “Collision-tolerant autonomous navigation through manhole-sized confined environments,” in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2020, pp. 84–89.

[47] S. Khattak, C. Papachristos, and K. Alexis, “Keyframe-based direct thermal-inertial odometry,” in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2019.

[48] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. Salakhutdinov, and Y. Bengio, “Architectural complexity measures of recurrent neural networks,” in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS’16, 2016, p. 1830–1838.