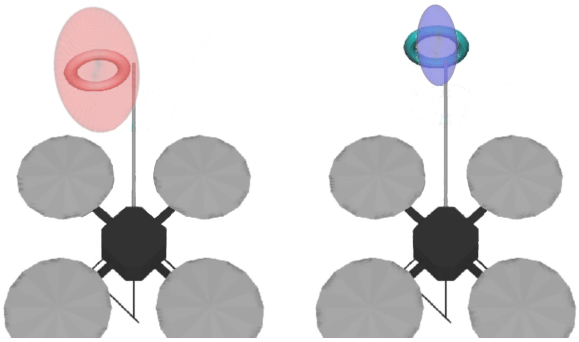
Learned Sensitivity Metrics for Robust and Accurate Motion Planning

Simon Wasiela, Marco Cognetti, Paolo Robuffo Giordano, Juan Cortés, Thierry Simeon

Learned Sensitivity Metrics for Robust and Accurate Motion Planning

Simon Wasiela[1], Marco Cognetti[1,2], Paolo Robuffo Giordano[3], Juan Cortés[1] and Thierry Siméon[1]

***Abstract*— This paper addresses the problem of computing robust trajectories against uncertainties in the robot model. Based on the notion of *closed-loop state sensitivity* introduced in [1], [2] for identifying deviations of the closed-loop trajectories of any robot/controller pair against variations of uncertain parameters in the robot model, uncertainty tubes can be derived for bounded parameter variations [3]. Such tubes were integrated within a motion planner named SAMP [4] to produce robust global plans, emphasizing the generation of low sensitivity trajectories. However, a bottleneck of this method remains the very high computational cost of the uncertainty tubes. In this paper, we mitigate this problem by proposing a novel framework that first incorporates a Long Short-Term Memory [5] (LSTM) neural network (NN) to provide fast and accurate uncertainty tubes estimation, and then reduces tracking errors at given points on the trajectory. We validate our framework on a full 3D-quadrotor UAV model performing a 'ring-recovering' task requiring high accuracy. Our results show the computational gain of the LSTM-based robust planning and that the subsequent optimization stage significantly improves the accuracy of the trajectory execution.**



(a) Missed ring pick (b) Successful ring pick

Fig. 1: Scenario with a drone that has to grasp a ring. (a) the uncertainty associated with its end-effector position is too large and the execution fails. (b) with a lower uncertainty the robot is able to successfully accomplish the task.

I. INTRODUCTION

Nowadays, motion planners can generate feasible and globally-optimal paths for high-dimensional systems and complex constraints/environments (e.g. see the recent survey [6]). However, most planners do not consider the unavoidable presence of uncertainties in the robot/environment model, and of a feedback action that will track the planned trajectories in presence of these uncertainties. This can often lead to a poor robustness and potential loss of optimality at runtime.

Efforts have been made to merge control and planning together. For example, the well known Model Predictive Control (MPC) technique [7] provides a "less local" control strategy. This technique consists in re-planning a local optimal trajectory taking into account a state feedback and a prediction of the evolution of the system state over a finite horizon. Nevertheless, MPC remains local in the vicinity of the reference trajectory. On the other hand, the global control-aware motion planner proposed in [8] relies on some coupling between planning and control, but uncertainties are not explicitly considered.

Other approaches based on contraction theory [9]–[13] or the so-called 'feedback motion planning' [14], [15] synthesize or rely on a specific robust controller that ensures that the trajectory remains within some uncertainty tubes, which can then be used in global planners as RRT [16]. These methods, however, remain limited to a specific form of control strategy and are often applicable only to specific class of systems for which the controller derivation is feasible in practice.

To overcome some of these problems, the sensitivity-aware motion planner (SAMP) [4] exploits the derivation of the uncertainty tubes in [3] (based on the so-called *closed-loop sensitivity matrix* [1], [2]) to generate globally robust trajectories for any controller and any robot dynamics. However, this planner suffers from the high computational cost of calculating the uncertainty tubes.

The contribution presented in this paper is twofold. We first propose a more efficient sampling-based motion planner (SBMP) relying on a Long Short-Term Memory (LSTM) [5] network that quickly and accurately estimates the uncertainty tubes and control input profiles along trajectories. In addition, the proposed framework locally optimizes the planned trajectory together with the controller gains in order to maximize the accuracy at a set of waypoints. Simulation results show the higher computational efficiency of the planning framework. Also, we tested our approach for the scenario shown in Fig.1 where a 3D-quadrotor with uncertain parameters has to perform an in-flight ring recovery task. The results confirm a better accuracy and robustness of the plan.

The paper is organized as follows. Sect. II first provides a reminder of the *closed-loop sensitivity* notions, and then describes our machine learning approach. Sect. III presents a framework to generate robust and accurate trajectories, and explains the incorporation of the LSTM-based neural network into a global planner. The dynamics of a 3D

* This work was supported by the project ANR-20-CE33-0003 "CAMP"

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France, {swasiela,simeon,mcognetti,jcortes}@laas.fr

[2] Université Toulouse III - Paul Sabatier, Toulouse, France.

[3] CNRS, Université de Rennes, Inria, IRISA, Rennes, France, prg@irisa.fr

quadrotor, the application of the learning method to the latter, and the planning scenario are described in Sect. IV. Simulations results are provided in Sect. V. Finally we draw some conclusions and future directions in Sect. VI.

II. LEARNING SENSITIVITY METRICS

A. Closed-loop sensitivity : Reminder

The notion of *closed-loop state sensitivity* [1], [2] quantifies deviations of the closed-loop trajectory of any robot/controller pair against variations of uncertain parameters in the robot model. Consider a generic robot dynamics

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}, \boldsymbol{p}),\ \boldsymbol{q}(t_0) = \boldsymbol{q}_0, \tag{1}$$

where $\boldsymbol{q} \in \mathbb{R}^{n_q}$ is the state vector, $\boldsymbol{u} \in \mathbb{R}^{n_u}$ the input vector, and $\boldsymbol{p} \in \mathbb{R}^{n_p}$ the vector containing (possibly uncertain) model parameters. Also assume presence of a tracking controller of any form to track a reference trajectory $\boldsymbol{r}_d(\boldsymbol{a}, t)$ parameterized by vector $\boldsymbol{a}$ s.t.,

$$\begin{cases} \dot{\boldsymbol{\xi}} = \boldsymbol{g}(\boldsymbol{\xi}, \boldsymbol{q}, \boldsymbol{a}, \boldsymbol{p}_c, \boldsymbol{k}_c, t),\ \boldsymbol{\xi}(t_0) = \boldsymbol{\xi}_0 \\ \boldsymbol{u} = \boldsymbol{h}(\boldsymbol{\xi}, \boldsymbol{q}, \boldsymbol{a}, \boldsymbol{p}_c, \boldsymbol{k}_c, t), \end{cases} \tag{2}$$

where $\boldsymbol{\xi} \in \mathbb{R}^{n_\xi}$ are the internal states of the controller (e.g., an integral action), $\boldsymbol{k}_c \in \mathbb{R}^{n_k}$ the controller gains and $\boldsymbol{p}_c \in \mathbb{R}^{n_p}$ the vector of nominal robot parameters used in the control loop.

In order to quantify how sensible the system is w.r.t. parameter variations, let the *state sensitivity matrix* $\boldsymbol{\Pi}$ be defined as (see [1])

$$\boldsymbol{\Pi}(t) = \left.\frac{\partial \boldsymbol{q}(t)}{\partial \boldsymbol{p}}\right|_{\boldsymbol{p}=\boldsymbol{p}_c} \in \mathbb{R}^{n_q \times n_p}, \tag{3}$$

and the *input sensitivity matrix* as (see [2])

$$\boldsymbol{\Theta}(t) = \left.\frac{\partial \boldsymbol{u}(t)}{\partial \boldsymbol{p}}\right|_{\boldsymbol{p}=\boldsymbol{p}_c} \in \mathbb{R}^{n_u \times n_p}. \tag{4}$$

These two matrices quantify, respectively, how sensible the states $\boldsymbol{q}(t)$ and the inputs $\boldsymbol{u}(t)$ are w.r.t. variations of $\boldsymbol{p}$ (w.r.t. the 'nominal' $\boldsymbol{p}_c$) for the closed-loop system (1–2). They do not have in general a closed-form expression, but it is possible to obtain a closed-expression for their dynamics. Therefore, the behavior of $\boldsymbol{\Pi}(t)$ and $\boldsymbol{\Theta}(t)$ is obtained in practice by numerical integration of a first-order differential equation along a planned trajectory.

The sensitivity framework has been extended in [3] in the case where each parameter p_i is assumed to vary within a range δp_i centered on the parameter nominal value p_{c_i} s.t. $p_i \in [p_{c_i} - \delta p_i, p_{c_i} + \delta p_i]$. In these conditions, the so-called *uncertainty tubes* can be computed as the envelope of the perturbed state/input trajectories obtained when the parameters vary in the given range. In order to compute these tubes, let $\boldsymbol{W} = \mathrm{diag}(\delta p_i^2)$ be a weighting matrix and

$$\Delta \boldsymbol{p}^T \boldsymbol{W}^{-1} \Delta \boldsymbol{p} = 1, \tag{5}$$

be an ellipsoid in parameter space centered at $\boldsymbol{p}_c$ and semi-axis δp_i, with $\Delta \boldsymbol{p} = \boldsymbol{p} - \boldsymbol{p}_c$. Assuming small variations of the parameters, one has $\Delta \boldsymbol{q} \approx \boldsymbol{\Pi}(t) \Delta \boldsymbol{p}$, and it is possible to obtain – see [3] for details – the corresponding ellipsoids in the state and the control spaces as

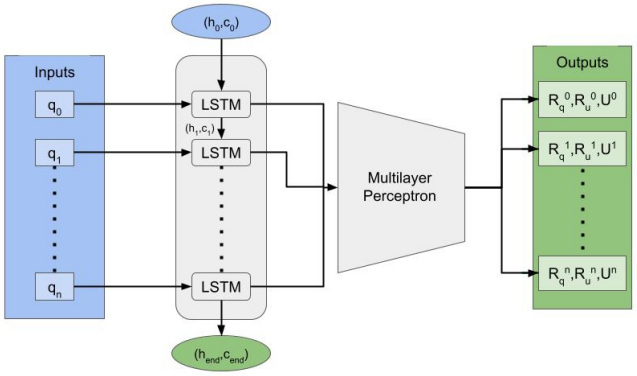


Fig. 2: A simplified architecture of the proposed LSTM NN. Blue blocks correspond to the inputs of the NN that are composed of a sequence of states $\boldsymbol{q}_k$ and an initial tuple (h_0, c_0). Green blocks refer to the outputs of the NN which are a sequence of radii and actuator's inputs $(\boldsymbol{R_q}^k, \boldsymbol{R_u}^k, \boldsymbol{U}^k)$, and the final tuple (h_{end}, c_{end}).

$$\Delta \boldsymbol{q}^T (\boldsymbol{\Pi} \boldsymbol{W} \boldsymbol{\Pi}^T)^{-1} \Delta \boldsymbol{q} = 1, \tag{6}$$

$$\Delta \boldsymbol{u}^T (\boldsymbol{\Theta} \boldsymbol{W} \boldsymbol{\Theta}^T)^{-1} \Delta \boldsymbol{u} = 1, \tag{7}$$

where $\Delta \boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_{nom}$, $\Delta \boldsymbol{u} = \boldsymbol{u} - \boldsymbol{u}_{nom}$, while $\boldsymbol{q}_{nom}$ (resp. $\boldsymbol{u}_{nom}$) is the state (resp. input) evolution of the closed-loop system in eqs. (1-2) in the nominal case ($\boldsymbol{p} = \boldsymbol{p}_c$). These ellipsoids represent the sets where the states and the inputs can vary in case the parameters vary in the specified ranges. Following the procedure given in [3], it is further possible to compute the radius $r_i(t)$ along the i-th component of the state that bounds its evolution over time. In order words,

$$q_{i,nom}(t) - r_i(t) \leq q_i(t) \leq q_{i,nom}(t) + r_i(t). \tag{8}$$

A similar derivation can also be done for the input ellipsoid. Such radii were used in [4] to perform robust collision checking within a motion planning framework. However, their computations rely on the knowledge of $\boldsymbol{\Pi}(t)$ and $\boldsymbol{\Theta}(t)$ that have been shown to be computationally expensive in the context of sampling-based motion planning, resulting in high planning times even for simple problems.

B. LSTM-based learning method

To leverage this computational issue, we propose a learning-based method to quickly and accurately estimate the tube radii in eq. (8) and the input profiles in eq. (2).

LSTM [5] is a recurrent neural network (NN) that has feedback connections encoding past events thanks to the so-called *hidden* and *cell* states. LSTM NN has already been used in recent years for motion planning applications, e.g., to quickly predict the surrounding state of the robot [17], [18], or to predict trajectories in sampling based algorithms [19]. Since LSTM NNs work very well in processing sequences of data – in particular time series – they perfectly fit with our framework, since the robot trajectory can be discretized

in some points (hereafter referred to as state) associated with some time instants, that will be treated as input sequences by the LSTM NN.

A simplified architecture of our NN is presented in Fig. 2, where the tuple (h_0, c_0) represents the initial hidden and cell states, respectively. The input sequence consists of a series of states, denoted with $\boldsymbol{q}_k$, evaluated along a trajectory at a given time step. The output is a sequence consisting of the inputs values $\boldsymbol{U}^k$ and of $\boldsymbol{R_q}^k$ and $\boldsymbol{R_u}^k$, that represent the radii of the uncertainty tubes, estimated at the *k*-th state of the trajectory, along the desired direction of the state and of the input spaces respectively. Finally, the tuple (h_{end}, c_{end}) corresponds to the hidden and the cell states at the last point of our sequence (i.e., the last state of our trajectory). The importance of such initial and final tuples is discussed in Sect. III-B. Training and prediction results are presented in Sect. V, as well as a discussion on the limitations.

III. ROBUST AND ACCURATE MOTION PLANNING VIA LEARNED SENSITIVITY METRICS

We first present the overall planning framework proposed to generate both robust and accurate trajectories in Sect. III-A. The two components of this framework are described next: Sect. III-B shows how to integrate the sensitivity learning method within a sampling-based motion planner, Sect. III-C then explains the accuracy optimization stage.

A. Robust and accurate planning framework (RA-SAMP)

In this section, we present *RA-SAMP*, a planning framework that generates robust and accurate trajectories. It works in two stages: (i) first, it generates a robust trajectory based on a Robust Sensitivity-Aware Motion Planner (*R-SAMP*) integrating the LSTM-based computation of the uncertainty tubes ; (ii) second, it optimizes the accuracy at some given points of this trajectory by minimizing the size of the uncertainty tubes at these locations. The motivation behind the second stage is to improve the accuracy of the planned robust trajectory for tasks – e.g., pick-and-place or insertion tasks – where the precision is important only at specific designed locations, as for picking the ring in Fig. 1.

We present *RA-SAMP* in its most general form in Alg.1. It takes as inputs the list of waypoints $list_{wpt} = (w_0, \ldots, w_n)$ for which the accuracy should be optimized, and the values of the nominal controller gains.

Algorithm 1 RA-SAMP $[list_{wpt}, gains_{init}]$

1: $\{\pi_{tot}, gains_{best}\} \leftarrow \{\emptyset, gains_{init}\}$;
2: **for** $(i = 1;\ i < len(list_{wpt});\ i = i + 1)$ **do**
3: $\quad \pi_i \leftarrow R\text{-}SAMP(list_{wpt}(i-1), list_{wpt}(i))$;
4: $\quad \pi_{tot} \leftarrow \pi_{tot} + \pi_i$;
5: **end for**
6: $\{\pi_{tot}, gains_{best}\} \leftarrow A\text{-}Optim(\pi_{tot}, gains_{best})$;
7: **return** $\{\pi_{tot}, gains_{best}\}$;

The first step of the algorithm consists in generating robust trajectories between successive waypoints in the list (line 3 of Alg. 1) by means of a SBMP framework called *R-SAMP* that utilizes our learning approach. These trajectories are concatenated into a global one, interconnecting all the waypoints (lines 2-5 in Alg. 1).

Then, the trajectory from *R-SAMP* is locally modified by *A-Optim* (line 6 in Alg. 1), an algorithm whose aim is to optimize the accuracy at given waypoints along the input trajectory. In particular, this algorithm samples, at each iteration, both the trajectory from R-SAMP and the controller's gains, modifying the former in order to minimize the uncertainty at the waypoints. The outputs of the algorithm are then: (i) a robust trajectory where the accuracy is maximized at the desired waypoints; (ii) the optimal controller gains.

Note that, even if *R-SAMP* benefits from the neural network, *A-Optim* cannot employ it because the controller gains are different from those used for training the network.

B. Robust sensitivity-aware motion planning (R-SAMP)

In this section, we describe how the learned uncertainty tubes can be incorporated in any tree-based SBMP in order to generate a robust sensitivity-aware motion planner (*R-SAMP*). As discussed before, one issue for computing the sensitivity metrics along a given trajectory comes from the possibly high computational time of the numerical integration of the dynamics for obtaining $\boldsymbol{\Pi}(t)$ and $\boldsymbol{\Theta}(t)$. This time might span from tens of milliseconds to several seconds – depending on the trajectory duration – and it sensibly increases the planning time for a SBMP. Moreover, an additional problem – that is specific when integrating the sensitivity machinery within a SBMP – comes from the fact that, for computing the sensitivity for a given node of a SBMP, the overall path for arriving at that node is needed.

However, this is not a problem for our framework thanks to the LSTM network, that naturally encodes this information within the hidden and the cell states. To this aim, these states are saved for each node of the tree built by the SBMP. Then, when expanding a node, the starting hidden and cell states (as for the tuple (h_0, c_0) of Fig.2) are already available from previous iterations, and the final ones (h_{end}, c_{end}) are embedded within the node generated by the expansion. This last tuple can then be reused in future extensions as the initial condition tuple (h_0, c_0). Note that such mechanism of using the tuples (h, c) limits its validity to tree-based planners (e.g., [20]–[24]), where each node has a single parent. Therefore, planners whose nodes may have multiple parents (e.g., PRM [25]) cannot employ such mechanism.

The LSTM network is used by the tree-based planner inside the function $RobustCC_{NN}$ whose pseudocode is reported in Alg.2. Its role is to validate the trajectory provided as input, together with the tuple (h_0, c_0) of the first trajectory state. The outputs of this function are a boolean value (*validity*) that specifies if the trajectory is valid or not, and the tuple (h_{end}, c_{end}), computed for the final state of the validated trajectory. The trajectory to be checked (π) is first discretized (π_d) using a time step ΔT (line 2). Then, all the radii and the inputs profiles $(\boldsymbol{R_q}, \boldsymbol{R_u}, \boldsymbol{U})$ on this trajectory are predicted by the LSTM network by means of the *SensiNN*

function (line 3 in Alg. 2). This function takes into account the initial trajectory conditions encoded in the tuple (h_0, c_0), and it returns the above-mentioned radii and inputs, together with the pair (h_{end}, c_{end}) by using the LSTM network.

For each state $\boldsymbol{q}_k$ of the discretized trajectory, the function *IsValid* (line 5 in Alg. 2) performs a robust collision checking, testing if there are no collisions despite the state uncertainty, and if the inputs are within some bounds that the system can exert. Since the first operation is known to be computationally expensive, the input bounds are tested first and, in case they are respected, the collisions are checked between the robot and the environment.

Algorithm 2 $RobustCC_{NN}$ $[\pi, (h_0, c_0)]$

1: $validity \leftarrow true$;
2: $\pi_d \leftarrow DiscretizedTraj(\pi, \Delta T)$;
3: $\{R_q, R_u, U, (h_{end}, c_{end})\} \leftarrow SensiNN(\pi_d, (h_0, c_0))$;
4: **for each** q_k **in** π_d **do**
5: **if not** $IsValid(R_u^k, U^k, R_q^k, q_k)$ **then**
6: $validity \leftarrow false$;
7: $break$;
8: **end if**
9: **end for**
10: **return** $\{validity, (h_{end}, c_{end})\}$;

Note that this robust collision checker can be employed within any tree-based planner. An example on how to include $RobustCC_{NN}$ for the case of a standard RRT planner is given in Alg. 3 with the pseudocode of the *R-SARRT* function, as a particular instance of *R-SAMP* planner. The same notation is used in Sect. V. As shown in lines 7-12 of Alg. 3, a valid output from $RobustCC_{NN}$ simply corresponds to a successful extension from a node, and the final state of the trajectory is inserted in the tree as a new node, storing in it also (h_{end}, c_{end}).

Algorithm 3 R-SARRT $[q_{init}, q_{goal}]$

1: $T \leftarrow \{q_{init}, q_{goal}\}$;
2: **for** k = 0 **to** K **do**
3: $q_{rand} \leftarrow Sample()$;
4: $q_{near} \leftarrow Nearest(T, q_{rand})$;
5: $\pi \leftarrow Steer(q_{near}, q_{rand})$;
6: $(h_0, c_0) \leftarrow q_{near}.getHidden()$;
7: $\{valid, (h_{end}, c_{end})\} \leftarrow RobustCC_{NN}(\pi, (h_0, c_0))$
8: **if** $valid$ **then**
9: $q_{rand}.setHidden(h_{end}, c_{end})$;
10: $T.addVertex(q_{rand})$;
11: $T.addEdge(q_{near}, q_{rand})$;
12: **end if**
13: **end for**
14: **return** T;

C. Accuracy optimization (A-Optim)

The choice to use a local optimization is justified by the cost function considered in order to optimize the accuracy at desired waypoints. Indeed, the latter is defined as:

$$c = \mathbb{E}[\boldsymbol{R_q}] + \mathbb{V}[\boldsymbol{R_q}] \tag{9}$$

which aims at minimizing the average size of the radii and their variance at each desired waypoints. The variance is considered in this cost function so that the minimization of a radius at a given point does not lead to a growth of another radius at another waypoint. This function is then neither additive (i.e considering two trajectories (π_1, π_2), the cost of their concatenation $c(\pi_1|\pi_2) \neq c(\pi_1) + c(\pi_2)$), nor monotonic, which makes it impossible to optimize in most global planners. This cost function is then optimized within the *A-Optim* method while other cost function can be optimized in the *R-SAMP* (e.g. path length). Given that we do not have the analytic form of the cost function derivatives and that estimating them would be expensive, the accuracy optimization has to be performed by a derivative-free (or black-box) method, e.g., the random shortcut algorithm [26].

IV. APPLICATION TO A 3D-QUADROTOR

In Sect.IV-A, we introduce the scenario on which we tested our framework, using a quadrotor as robotic platform. Then, in Sect. IV-B, we explain how we learned the uncertainty tubes for the quadrotor case.

A. 3D-quadrotor application and planning scenario

The dynamic model of the robot is the same full 3D quadrotor model as described in [3]. The tracking controller considered in this work is the so-called Lee (or geometric) controller [27] where the control inputs are the squared rotor speeds $\boldsymbol{u} = [\omega_1^2\, \omega_2^2\, \omega_3^2\, \omega_4^2]^T$.

An in-flight ring recovery task is considered for testing our accurate optimization framework. In particular, the quadrotor is tasked to recover some rings by means of a perch mounted along its x-axis, as depicted in Fig. 1. The task is performed in the environment shown in Fig. 3, in which the robot must first recover 3 rings in the departure room (red rectangle in the figure), then it must pass through a window of dimension $1.5m \times 1m$ (blue rectangle) for reaching the second room (green rectangle) and collecting other 4 rings. Finally, it has to return to the departure room, passing again through the small window. In Fig. 3, each ring is represented by a black dot, it has an inner radius of $0.06m$ and a mass of $50g$.

For this scenario we consider the uncertain parameters $\boldsymbol{p} = [m\, g_x\, I_x\, I_y\, I_z]^T$ where m denotes the mass of the system, g_x the position of its center of mass along the x-axis of the body frame, and I_x, I_y, I_z, are the principal moments of inertia in the body frame. Recovered rings will remain attached to the perch, modifying the mass of the quadrotor along its x-axis. This justifies why we considered as uncertain g_x only and not g_y, due to the fact that the mass and its distribution becomes mostly uncertain along the x-axis of the UAV. We chose our nominal parameters as $\boldsymbol{p}_c = [1.5, 0.0, 0.01, 0.01, 0.02]^T$ and their associated uncertainty range $\delta\boldsymbol{p} = [10\%, 5cm, 5\%, 5\%, 5\%]^T$ which represents the percentage variation of the parameters w.r.t. their associated nominal values except for g_x whose nominal

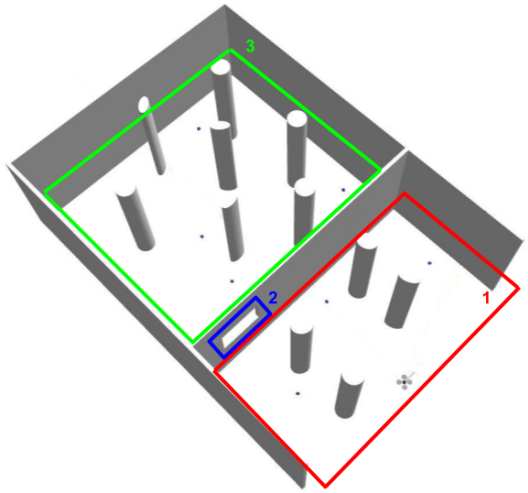


Fig. 3: Environment of the 'ring recovery task' scenario. The position of the 7 rings is represented by a black dot. The departure room of the drone (red rectangle) contains 3 rings and is connected by a small window (blue rectangle) to a second room room with 4 rings.

value is null. The quadrotor trajectories are planned using the *Kinosplines* steering method of [28], and a RRT^* algorithm [20] minimizing the time-length. Robust collision checking is performed by simply considering a bounding sphere of the UAV increased by the predicted state uncertainty radii $\boldsymbol{R_q}$.

B. Learning uncertainty tubes for the 3D-quadrotor

As mentioned in Sect. II-B, our LSTM-based network predicts the uncertainty tubes' radii and the inputs of the system. In the quadrotor case, the control inputs to be predicted are $\boldsymbol{U} = [U_1, U_2, U_3, U_4]^T$, with $U_i = \omega_i^2$. Regarding the uncertainty tubes, the LSTM-based network predicts $\boldsymbol{R_q} = [R_x, R_y, R_z]^T$ (with $R_k, k \in \{x, y, z\}$ the radius of the state uncertainty tube along the $\{x, y, z\}$-axis) and $\boldsymbol{R_u} = [R_{u1}, R_{u2}, R_{u3}, R_{u4}]^T$ (with R_{ui} the radius of the input uncertainty tube associated with the i-th control input of the system).

Then, as shown in eq.(6), the computation of these radii is strongly related to the one of the *state/input sensitivity matrices* (3–4). Such matrices can be computed numerically by forward integration and thus strongly rely on the Jacobian of the system dynamics. This justifies our choice of using the global velocities and accelerations as well as the yaw angle (Ψ) along the trajectory to be the components of $q_i = [cos(\Psi)\ sin(\Psi)\ \dot{q}_d\ \ddot{q}_d]^T$ evaluated at time $i\Delta T$, where $q_d = [x\ y\ z\ \Psi]^T$ and ΔT is the same time step that will be used for the collision checking by the planner.

Finally, the LSTM-based network is trained using the MSE criterion on 10.000 randomly generated trajectories with an execution time of 15s. In order to evaluate the accuracy of the network, we generated an evaluation set composed of 1.000 trajectories whose duration is slightly longer (17s) w.r.t. the ones used in the training set. The performance of the learning is illustrated in Table I that reports the accuracy for each radius according to the L1 error norm.

	R_x	R_y	R_z	U_i	R_{u1}, R_{u3}	R_{u2}, R_{u4}
Accuracy	**0.96**	**0.95**	**0.91**	**0.99**	**0.99**	**0.85**

TABLE I: Accuracy according to L1 norm of the relative error along each component of the predicted vector. R_{ui} is the radius of the uncertainty tube associated with the i-th input and $U_i, i \in \{1, 2, 3, 4\}$ represents the i-th actuator input

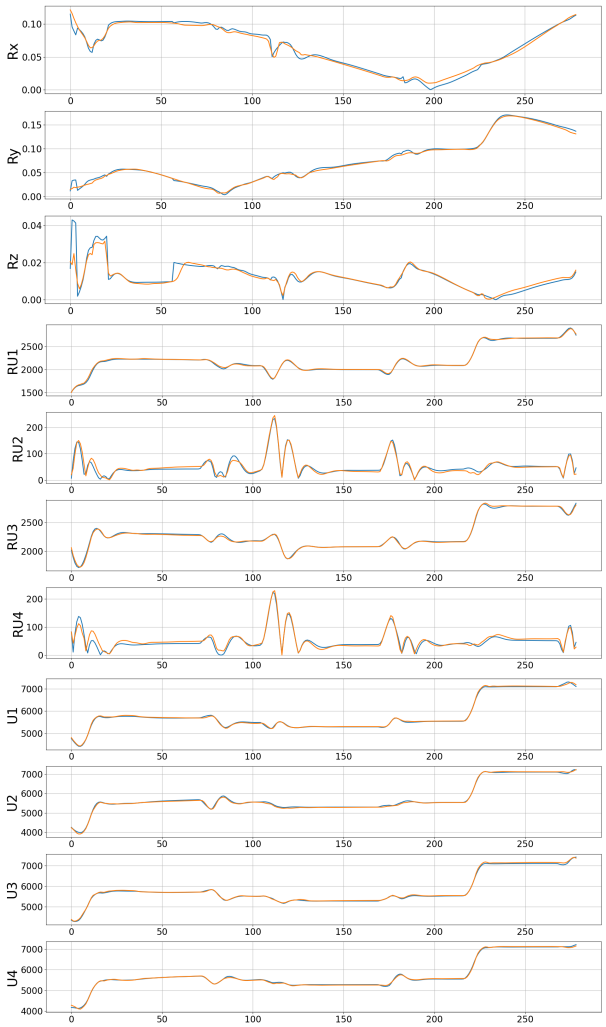


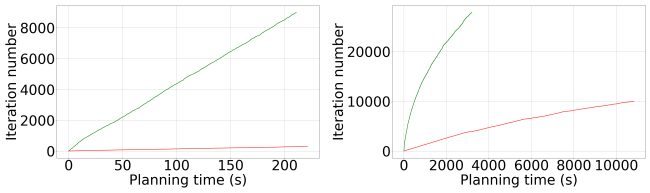
Fig. 4: Example of network predictions along a trajectory (orange) against true values (blue). $R_{x,y,z}$ are expressed in m, and input associated values $(R_{ui}, U_i)_{i=1,4}$ in rpm.

V. SIMULATION RESULTS

This section first presents results showing the quality of the learning-based computation uncertainty tubes and its high efficiency when used for robust motion planning. Then, we illustrate the ability of the proposed *RA-SAMP* planning approach to generate high accuracy trajectories for the scenario illustrated by Figure 3.

A. Performance of the learning-based tube computation

Figure 4 compares the learning-based prediction of the tubes radii and inputs wrt. their real values. The curves illustrate the excellent accuracy on the predictions of the



(a) RRT (b) RRT^*

Fig. 5: Number of (a) RRT / (b) RRT^* iterations as a function of planning time using the learning method (green) or the $\mathbf{\Pi}$ dynamics integration (red), as done in [4].

inputs $\boldsymbol{U}$ as well as on the radii R_{u1}, R_{u3}, R_x, R_y while a lower accuracy is observed on R_{u2}, R_{u4}, R_z. However, note that the absolute values of these radii are much smaller compared to the others. In other words, the relative accuracy (or, equivalently, the relative error, normalized by the reference value) is comparable and similar in all the radii, confirming the validity of the prediction of the LSTM-based network. Moreover, in order to guarantee the prediction robustness, we add for each component of the predicted vector the quartile of its relative error observed on the evaluation set.

Fig. 5 shows the significant performance improvement of using this learning-based prediction within a SBMP for checking the robustness of the local tree expansions using the $RobustCC_{NN}$ function (see Alg. 3), against the previous version [4] integrating the $\mathbf{\Pi}$ dynamics. Results provided for RRT and RRT^* compare the number of iterations of the main loop of the algorithm a function of computing time, showing for both cases a significant gain thanks to the proposed learning method. Note that in the RRT case this time gain is constant (45 times faster) because the expansion benefits from the neural network only once per iteration. In the RRT^* case, the denser the tree, the more robust collision tests are required for the rewiring connection phase. Therefore, more time is saved by using the learning method. In particular, the gain on the planning time can reach more than two-orders of magnitudes for problems requiring a significant amount of iterations.

Note that the application of the learning method is limited to trajectories having a shorter execution time than those used for training, which constraints the time-length of the local paths used by the planner for the tree expansions. Also, the predictions are only valid for the variation of the parameters $\delta\boldsymbol{p}$ chosen during the generation of the training set, and the model is trained for given values of the controller gains.

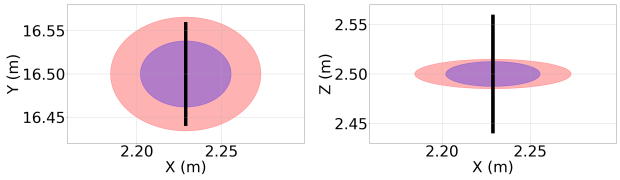
B. Accuracy optimization evaluation

Finally, we demonstrate the efficiency of our accuracy optimization framework on the in-flight ring recovery task scenario by planning trajectories that minimize the uncertainties for the end effector position of the perch at the location of the rings. The rings that are collected by the robot's perch modify the quadrotor mass at run time. The planned trajectories are executed in the Gazebo simulator.

For this experiment, twenty trajectories are planned, using the classic RRT^*, a robust version of the latter ($R\text{-}SARRT^*$), and a robust version also optimizing the accuracy ($RA\text{-}SARRT^*$), where the *A-Optim* function is a simple robust variant of the shortcut algorithm [26] in which the controller gains are also sampled between 50% and 150% of their nominal values. Table II shows the average execution failure[1] percentage for each type of trajectory, as well as the average number of rings recovered. We can see that LSTM-based tubes prediction allows to generate more robust trajectories than the standard RRT^*, and that the accuracy optimization allows to recover even more rings. Finally, Fig.6 shows an example of uncertainty ellipsoids before and after the accuracy optimization (i.e., $R\text{-}SARRT^*$ vs. $RA\text{-}SARRT^*$) at one ring location. In particular, the uncertainty, that was larger than the radius of the ring, is reduced and becomes smaller after optimization.

	RRT^*	$R\text{-}SARRT^*$	$RA\text{-}SARRT^*$
Failure (%)	**65.0**	**5.0**	**5.0**
Number of rings picked	**1.2** ± 1.6	**3.1** ± 0.9	**4.1** ± 1.1

TABLE II: Average number of failures, together with the average number of rings picked over 20 simulations.



(a) XY uncertainty (b) XZ uncertainty

Fig. 6: Example of uncertainty ellipsoid at the first ring location, before (pink) and after (purple) accuracy optimization, displayed along the XY plane (left) and XZ plane (right). The ring shape is displayed as a black rectangle. Similar results are observed for the other rings.

VI. CONCLUSION

In this paper, we propose a motion planner able to generate trajectories that are intrinsically robust against parametric uncertainties for any robot/controller pair. It is hinged on a LSTM-based learning approach that quickly and accurately estimates the control inputs and the uncertainty tubes of the state and of the inputs. The results confirm the efficiency of the proposed learning method and highlight the impact of integrating it within a motion planner, resulting in a significant reduction of the planning times. Moreover, we showed that our framework is able to locally optimize the planned trajectory in order to minimize the size of the uncertainty tubes of the state at some desired locations, allowing the system to accurately perform a precision task. Future works will focus on considering uncertainties not only at the parametric level, by extending the computation of the tubes for external disturbances. Finally, we aim to experimentally validate the proposed framework on a real robotic platform.

[1] An execution failure is characterized by zero ring recovered.

REFERENCES

[1] P. Robuffo Giordano, Q. Delamare, and A. Franchi, "Trajectory generation for minimum closed-loop state sensitivity," in *2018 IEEE Int. Conf. on Robotics and Automation*, 2018, pp. 286—-293.

[2] P. Brault, Q. Delamare, and P. Robuffo Giordano, "Robust trajectory planning with parametric uncertainties," in *2021 IEEE International Conference on Robotics and Automation*, 2021, pp. 11 095–11 101.

[3] P. Brault and P. Robuffo Giordano, "Tube-based trajectory optimization for robots with parametric uncertainty," http://rainbow-doc.irisa.fr/pdf/tube_based_traj_opt.pdf.

[4] S. Wasiela, P. Robuffo Giordano, J. Cortés, and T. Simeon, "A sensitivity-aware motion planner (samp) to generate intrinsically-robust trajectories," in *2023 IEEE International Conference on Robotics and Automation*, 2023.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] J. D. Gammel and M. P. Strub, "A survey of asymptotically optimal sampling-based motion planning methods," *Annu. Rev. Control Robot. Auton. Syst.*, vol. 4, no. 1, 2021.

[7] B. Houska and M. E. Villanueva, "Robust optimization for mpc," in *Handbook of model predictive control*. Springer, 2019, pp. 413–443.

[8] M. Tognon, E. Cataldi, H. T. Chavez, G. Antonelli, J. Cortés, , and A. Franchi, "Control-aware motion planning for task-constrained aerial manipulation," *IEEE Robotics and Automation Letters*, vol. 3, no. 13, pp. 2478–2484, 2018.

[9] Z. Manchester and S. Kuindersma, "Robust direct trajectory optimization using approximate invariant funnels," *Autonomous Robots*, vol. 43, pp. 375–387, 2019.

[10] G. Chou, N. Ozay, and D. Berenson, "Model error propagation via learned contraction metrics for safe feedback motion planning of unknown systems," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 3576–3583.

[11] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5883–5890.

[12] P. Zhao, A. Lakshmanan, K. Ackerman, A. Gahlawat, M. Pavone, and N. Hovakimyan, "Tube-certified trajectory tracking for nonlinear systems with robust control contraction metrics," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5528–5535, 2022.

[13] S. Singh, H. Tsukamoto, B. T. Lopez, S.-J. Chung, and J.-J. Slotine, "Safe motion planning with tubes and contraction metrics," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 2943–2948.

[14] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[15] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[16] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[17] Y. Jeong, S. Kim, and K. Yi, "Surround vehicle motion prediction using lstm-rnn for motion planning of autonomous vehicles at multi-lane turn intersections," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 1, pp. 2–14, 2020.

[18] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[19] R. S. Nair and P. Supriya, "Robotic path planning using recurrent neural networks," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2020, pp. 1–5.

[20] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[21] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[22] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

[23] D. Devaurs, T. Siméon, and J. Cortés, "Optimal path planning in complex cost spaces with sampling-based algorithms," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 415–424, 2015.

[24] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, 2010.

[25] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[26] R. Geraerts and M. Ovemars, "Creating high-quality paths for motion planning," *The International journal of robotics research*, vol. 26, 2007.

[27] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *49th IEEE Conference on Decision and Control (CDC)*, December 2010, p. 5420–5425.

[28] A. Boeuf, J. Cortés, and T. Simeon, "Motion planning," in *Aerial Robotic Manipulation*. Springer, 2019, pp. 317–332.