Robust Motion Planning with Accuracy Optimization based on Learned Sensitivity Metrics

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

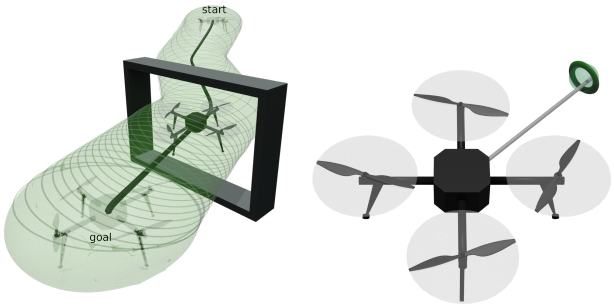
***Abstract*— This work addresses the problem of generating robust and accurate trajectories taking into account uncertainties in the robot dynamic model. Based on the notion of *closed-loop sensitivity*, which quantifies deviations in the closed-loop trajectories of any robot/controller pair against uncertainties in the robot model parameters, uncertainty tubes can be derived for bounded parameter variations. In our prior work, such tubes were integrated within a motion planner named SAMP to produce robust global plans, emphasizing the generation of trajectories with low sensitivity to model uncertainty. However, the high computational cost of the uncertainty tubes is a bottleneck for this method. Here, we solve this problem by proposing a novel framework that first incorporates a Gated Recurrent Unit (GRU) neural network to provide fast and accurate estimation of uncertainty tubes and then minimizes these tubes at given points along the trajectory. We experimentally validate our framework on a 3D quadrotor in two challenging scenarios: a navigation through a narrow window, and an in-flight "ring catching" task that requires high accuracy.**

***Index Terms*— Motion and Path Planning, Planning under Uncertainty, Integrated Planning and Control**

I. INTRODUCTION

Robust motion planning for dynamic systems to ensure the safe execution of the planned trajectory in the presence of parametric uncertainties is an important but challenging problem. To this end, several recent approaches [1]–[7] (see Sect.II) rely on so-called "uncertainty tubes" that bound the state evolution of the system given a model of uncertainty. However, these approaches are either limited to specific system classes, require the design of specific robust controllers, or only address the presence of external disturbances, neglecting model parametric uncertainty. Other approaches propose more general probabilistic strategies propagating the system dynamics for different random uncertainties to estimate the set of states that can be reached by the system [8], [9]. However, a good estimate of this set may require sampling a large number of states, which can quickly become computationally heavy for complex systems and controllers.

To overcome some of the aforementioned problems, the sensitivity-aware motion planner (SAMP) proposed in [10] exploits the derivation of uncertainty tubes of [11], based on the so-called *closed-loop sensitivity* [12], [13]. This approach is applicable to any controller and robot model, taking into account parametric uncertainties. However, this planner suffers from the high computational cost of computing the uncertainty tubes. Moreover, SAMP and the aforementioned methods focus on computing robust trajectories but do not consider the problem of *also* minimizing uncertainty tubes at a desired location for increasing the accuracy of specific tasks (e.g., insertion, grasping).



(a) Robust motion planning (b) Accuracy optimization

Fig. 1: Two scenarios considered for the experimental validation of the proposed method: (a) Robust navigation of a drone through a narrow window. (b) Precision in-flight 'ring catching' task, where the uncertainty on the position of the perch end-effector is minimized to successfully accomplish the task. A video of the experiments is available at: https://laas.hal.science/hal-04642257

With respect to these considerations, the contribution of this paper is twofold: (i) We propose a computationally efficient version of the SAMP algorithm, relying on a Gated Recurrent Unit neural network (GRU) [14] (see Sect. III), which quickly and accurately estimates time-varying uncertainty tubes and control input profiles along trajectories. We present a general way of incorporating this type of network into a sampling-based tree planner (see Sect. IV) in order to predict the uncertainty tubes and control inputs along trajectories. (ii) On the basis of this new planner, we propose a comprehensive framework that not only plans robust trajectories but also locally optimizes them together with the controller gains to maximize accuracy at some desired states of the planned trajectory for any system/controller.

Fig. 1 illustrates the application to a quadrotor model (see Sect.V) in two different experimental scenarios involving uncertain parameters affecting the robot model (see Sect.VI): (i) a robust navigation through a narrow window; (ii) an in-flight ring-catching task demonstrating the robustness and accuracy of the proposed framework.

* This work was supported by the project ANR-20-CE33-0003 "CAMP" and by the Chaire de Professeur Junior grant no. ANR-22-CPJ1-0064-01.

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

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

II. RELATED WORK

A common way for guaranteeing the robustness of planned motions is to use "uncertainty tubes" that bound the state of the system in the presence of disturbances/uncertainties. These tubes can be computed in different ways and with different assumptions about the system/controller.

For instance, a robust version of the well-known Model Predictive Control (MPC), which is an online planning technique, was proposed in [1] where a feedback tracking controller is used to maintain the system state within a consistent "tube" around the nominal MPC trajectory in the presence of disturbances. Nevertheless, MPC remains local in the vicinity of a reference trajectory.

A more global approach as the so-called 'feedback motion planning' [2] builds a tree of local Linear Quadratic Regulator (LQR) feedback controllers for which the envelope of a Lyapunov function is computed. Nevertheless, this approach is constrained to the particular class of LQR controllers, and the computation of the tube is time-consuming. Based on this previous work, the offline computation of a 'tube library' was proposed in [3]. Although this allows robust planning at runtime, the solutions are constrained by a predetermined number of trajectories.

In [4]–[6], tubes were computed by leveraging the so-called 'control contraction metrics'. These contraction metrics represent a generalization of control Lyapunov functions and are used to synthesize specific robust controllers for which tubes can be derived according to their associated contraction rate and given maximum disturbance limits. Consequently, these methods remain limited to the use of specific controllers, which may not always be amenable to a reasonably simple implementation in real-world cases.

In the FaSTrack framework [7], a simplified system dynamic model, that does not allow parametric uncertainties to be taken into account, was used to compute invariant tubes that are used at runtime. Furthermore, to ensure that the system actually stays inside these tubes, a specific synthesized optimal tracking controller was used.

While the aforementioned methods focus on generating robust trajectories to external disturbances or unmodeled forces (e.g., wind, friction), none of them considers potential mismatches or fluctuations in the robot model parameters during runtime, such as changes in mass or displacements of the center of mass. Moreover, many of these methods hold for a specific class of systems or a specific synthesized controller and, thus, lack generality.

To tackle some of these issues, the RandUp-RRT proposed in [8], [9] is applicable to any system and controller. The idea is to estimate for each node of the tree the set of states that can be reached by the system using 'particles', which corresponds to a dynamic propagation of the system in the presence of random parameters uncertainty. Furthermore, these sets can be approximated for any system and controller. However, the guarantee that these reachable sets are conservative relies on additional padding or a large number of particles, and the more particles considered, the longer the computation time will be compared to conventional algorithms. Nevertheless, this work is the closest to ours and will be used as a baseline for comparison.

III. LEARNING SENSITIVITY METRICS

A. Closed-loop sensitivity: A reminder

The notion of *closed-loop sensitivity* was introduced in [12], [13] for quantifying how variations of some model parameters (supposed to be uncertain) affect the evolution of the system in closed-loop, i.e., by also taking into account any controller chosen for executing the task. Consider a generic robot dynamics

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

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

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

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 parameters used in the control loop.

In order to quantify how sensitive 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), the *state sensitivity matrix* $\boldsymbol{\Pi}$ and the *input sensitivity matrix* $\boldsymbol{\Theta}$ are defined in [12] and [13], respectively. They do not have in general a closed-form expression but their evolutions in time can be computed according to their following dynamics (see [12], [13] for more details):

$$\begin{cases} \dot{\boldsymbol{\Pi}}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\boldsymbol{\Pi} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\boldsymbol{\Theta} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}}, \\ \dot{\boldsymbol{\Pi}}_\xi(t) = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}\boldsymbol{\Pi} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\xi}}\boldsymbol{\Pi}_\xi, \\ \boldsymbol{\Theta}(t) = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}}\boldsymbol{\Pi} + \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{\xi}}\boldsymbol{\Pi}_\xi. \end{cases} \tag{3}$$

Given a bounded range δp_i for each uncertain parameter p_i s.t. $p_i \in [p_{c_i} - \delta p_i, p_{c_i} + \delta p_i]$, and assuming small variations of the parameters s.t. $\Delta \boldsymbol{q} \approx \boldsymbol{\Pi}(t)\Delta \boldsymbol{p}$, it is possible to obtain the so-called *uncertainty tubes*. The tube along the i-th component of the state is characterized by its radius $r_{q,i}(t)$, which bounds the state evolution over time. In other words,

$$q_{n,i}(t) - r_{q,i}(t) \leq q_i(t) \leq q_{n,i}(t) + r_{q,i}(t), \tag{4}$$

with $r_{q,i}(t) = \sqrt{\boldsymbol{n_i}^T \boldsymbol{K}_{\boldsymbol{\Pi}}(t)\boldsymbol{n_i}}$, $\boldsymbol{K}_{\boldsymbol{\Pi}}(t) = \boldsymbol{\Pi}(t)\boldsymbol{W}\boldsymbol{\Pi}(t)^T$, where $\boldsymbol{n_i}$ is the i-th dimension of the state, and $\boldsymbol{W}$ is a diagonal matrix where its elements are the components of $\delta \boldsymbol{p}$ (see [11] for details). Note that such bounds apply to the *nominal state* ($\boldsymbol{q_n}$)[1] and that similar tubes can be obtained for the control inputs.

[1] A nominal state refers to the simulated state of the system in closed-loop under nominal system parameters (i.e., $\boldsymbol{p} = \boldsymbol{p}_c$).

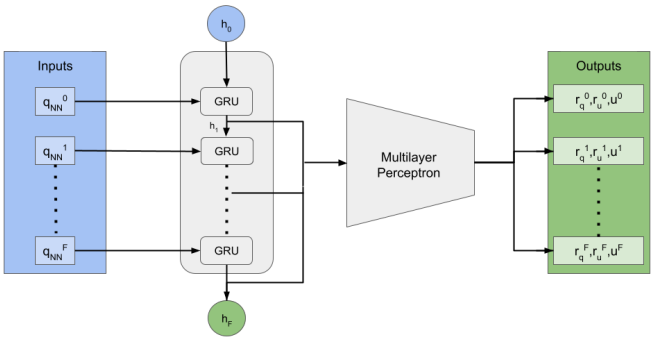


Fig. 2: Representation of the GRU architecture. Blue blocks correspond to the inputs composed of a sequence of states $\boldsymbol{q_{NN}^k}$ and an initial hidden state (h_0). Green blocks refer to the output, which is a sequence of radii and control inputs ($\boldsymbol{r_q}^k, \boldsymbol{r_u}^k, \boldsymbol{u}^k$), and the final hidden state (h_F).

Since the uncertainty tubes' radii depend on the state sensitivity $\mathbf{\Pi}$, we need to numerically integrate eq. (3) to compute them. Depending on the number of parameters and the dimension of the system state, this integration may be computationally expensive. For example, referring to our quadrotor case in Sect.V, this requires solving more than a hundred ordinary differential equations. For trajectories composed of a hundred samples, this may require from tens to hundreds of milliseconds. These tubes were used in [10] to perform robust collision checking within a motion planning framework. However, each iteration of the planner requires a new tube computation which results in prohibitively high planning times even for simple problems.

B. GRU-based learning method

To leverage the computational issue raised by integrating the sensitivity dynamics in (3), we propose a one layer GRU-based method to quickly and accurately estimate the tube radii ($\boldsymbol{r_q}$ and $\boldsymbol{r_u}$) and the input $\boldsymbol{u}$ profiles.

The use of recurrent networks for motion planning applications has rapidly grown in recent years, e.g., to quickly predict the surrounding state of a robot [15], or to predict trajectories in sampling-based algorithms [16]. They rely on feedback connections that encode past events through the so-called *hidden* states. Since they work very well in processing sequences of data – in particular time series – they perfectly fit with our framework, since the planned trajectory can be discretized into points at a given time step (hereafter referred to as *desired states*), that will be treated as input sequences by the network. Among the most effective are GRU [14] and LSTM [17], which process the 'memory' of the hidden state more efficiently. We use GRU in this work as it provides more advantages such as memory saving (see [18]).

A simplified representation of our GRU architecture is presented in Fig. 2, where h_0 represents the initial hidden state and the input sequence consists of a series of states, denoted by $\boldsymbol{q}_{NN}^k$, evaluated at the k-th time step of a desired trajectory. The output is a sequence consisting of the nominal control input values $\boldsymbol{u}^k$ and of $\boldsymbol{r_q}^k$ and $\boldsymbol{r_u}^k$, which represent the radii of the uncertainty tubes, estimated at the k-th state of the trajectory, along the desired directions of the state and of the input spaces, respectively. Finally, h_F corresponds to the hidden state at the last point of the sequence (i.e., the last state of the trajectory). The importance of the initial and final hidden states is discussed in Sect.IV-B. An application of this machine learning approach to a quadrotor is presented in Sect.V, together with the training procedure and an evaluation of its performance.

IV. ROBUST MOTION PLANNING WITH ACCURACY OPTIMIZATION VIA LEARNED SENSITIVITY METRICS

Sect. IV-A provides an overview of our planning framework to generate robust and accurate trajectories. Sect. IV-B shows how to integrate the sensitivity learning method within a sampling-based motion planner, while Sect. IV-C explains the accuracy optimization stage.

A. Robust and accurate planning framework

The method consists of two stages: (i) first, it generates a robust trajectory based on a Robust Sensitivity-Aware Motion Planner (R-SAMP) – explained in Sect. IV-B – that utilizes the GRU-based computation of the uncertainty tubes; (ii) second, it optimizes the accuracy at some given states along 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 minimizing the deviation from the nominal trajectory is important only at specific designed locations, as for picking the ring in Fig. 1.

The pseudo-code of RA-SAMP (Robust and Accurate Sensitivity-Aware Motion Planner) is presented in Alg.1. It takes as input (line 1) a list of desired states $list_d = (q_d^0, \ldots, q_d^n)$ for which the accuracy should be optimized, and the initial controller gains vector k_c^{init} considered constant all along the trajectory.

Algorithm 1 RA-SAMP $[list_d, k_c^{init}]$

1: $\pi_d^{tot} \leftarrow \emptyset$;
2: **for** $(i = 1;\ i < len(list_d);\ i = i + 1)$ **do**
3: $\quad \pi_d^{tot} \leftarrow \pi_d^{tot}$+R-SAMP$(list_d(i-1), list_d(i))$;
4: **end for**
5: $\{\pi_d^{tot}, k_c^{opt}\} \leftarrow$A-Optim$(list_d, \pi_d^{tot}, k_c^{init})$;
6: **return** $\{\pi_d^{tot}, k_c^{opt}\}$;

The first step of the algorithm consists of generating robust trajectories (π_d^i) between successive desired states in the list (line 3 of Alg. 1) by means of a robust sensitivity-aware motion planner called R-SAMP (explained in Sect. IV-B) that uses our learning approach. These trajectories are concatenated into a global one π_d^{tot}, connecting all the desired states of $list_d$ (lines 2-5 in Alg. 1).

The trajectory from R-SAMP is locally modified by A-Optim (line 6 in Alg. 1), aiming at optimizing the accuracy at specific desired states along the trajectory. This algorithm

iteratively samples both the trajectory from R-SAMP and the controller gains, adjusting the former to minimize uncertainty at these states. Indeed, as demonstrated in [19], optimizing both factors concurrently results in minimizing the uncertainty. The algorithm produces two offline outputs: (i) a robust desired trajectory π_d^{tot} optimized for accuracy at the desired states, and (ii) the optimized controller gains vector k_c^{opt}, considered constant throughout the trajectory.

B. Robust Sensitivity-Aware Motion Planner

This section explains how the learned uncertainty tubes can be incorporated into any sampling-based tree planner in order to obtain a robust sensitivity-aware motion planner (R-SAMP). As highlighted before, a key challenge in computing such tubes for a given trajectory lies in the high computational cost of numerically integrating the dynamics of $\mathbf{\Pi}(t)$ and $\mathbf{\Theta}(t)$. Additionally, when extending the tree and computing these sensitivity matrices, various initial conditions (e.g. initial control input, $\mathbf{\Pi}_0$, etc.) must be embedded in the tree nodes.

We solve this problem thanks to the GRU network, which naturally encodes this information in its "memory" terms, i.e., the so-called hidden state (see [14]). An interesting feature of the algorithm is to leverage this latent state to embed the initial conditions into each node. This enables the reuse of the updated initial conditions for predictions in future extensions. Note that a hidden state h is unique according to its parent. Therefore, its use is only applicable to tree-based planners, where each node has a single parent. We show in Alg. 2 how to incorporate this hidden state and tube predictions for the case of a standard RRT planner [20] with the pseudocode of the R-SARRT algorithm, as a particular instance of an R-SAMP planner. Note that the use of this hidden state and tube predictions can be similarly applied to other tree-based planners. For instance, in the results presented in Sect. VI-B, we used a robust RRT* implementation denoted R-SARRT*.

First, R-SARRT performs the standard RRT procedure (lines 1-5) that produces a local desired trajectory π_d between a sampled state (q^{rand}) and its nearest state (q^{near}) in the tree. Then, as the tubes are only valid around the nominal trajectories, which may differ from the desired ones depending on the controller performance, the nominal trajectory (π_n) is computed by the SimulateExecution function (line 6). It corresponds to the simulated tracking in closed-loop of the desired trajectory (π_d) to be robustly checked by the system under the nominal parameters (i.e. $\boldsymbol{p} = \boldsymbol{p}_c$). Next, the starting hidden state (as for h_0 of Fig. 2) is recovered from the tree node (line 7). Such initial condition is used together with the above-mentioned nominal trajectory by the GRU that returns all the radii and the control inputs profiles ($\boldsymbol{r_q}, \boldsymbol{r_u}, \boldsymbol{u}$), together with the final hidden state h_F to be reused as initial condition in subsequent iterations (line 8). Then, for each state of the nominal trajectory π_n, the function *IsRobust* (line 9) performs a robust collision checking by using the uncertainty radii, and tests if the inputs are within the admissible bounds that the system can exert. If

Algorithm 2 R-SARRT $[q^{init}, q^{goal}]$

1: $T \leftarrow$ InitTree(q^{init});
2: **while not** StopCondition(T, q^{goal}) **do**
3: $q^{rand} \leftarrow$Sample();
4: $q^{near} \leftarrow$ Nearest(T, q^{rand});
5: $\pi_d \leftarrow$ Steer(q^{near}, q^{rand});
6: $\pi_n \leftarrow$SimulateExecution(π_d);
7: $h_0 \leftarrow$GetNodeConditions(q^{near});
8: $\{\boldsymbol{r_q}, \boldsymbol{r_u}, \boldsymbol{u}, h_F\} \leftarrow$GRU($\pi_d, h_0$);
9: **if** $IsRobust(\boldsymbol{r_q}, \boldsymbol{r_u}, \boldsymbol{u}, \pi_n)$ **then**
10: SetNodeConditions(q^{rand}, h_F);
11: AddNewNode(T, q^{rand});
12: AddNewEdge(T, q^{near}, q^{rand});
13: **end if**
14: **end while**
15: **return** GetTrajectory(T, q^{init}, q^{goal});

the extension is valid, the final state of the desired trajectory is inserted in the tree as a new node, embedding at the same time the final hidden state h_F to be reused as initial condition in next iterations (line 10-12). Finally, the algorithm returns a global trajectory connecting q^{init} and q^{goal} if one exists in the tree (lines 15).

C. Accuracy optimization (A-Optim)

The application of a local optimization method at this level is justified by the cost function considered in order to optimize the accuracy at desired states. Indeed, the cost of a trajectory π is defined as:

$$c(\pi) = w_1\mathbb{E}[L] + w_2\mathbb{V}[L], \quad L = [\lambda_0 ... \lambda_n] \tag{5}$$

with $\mathbb{E}[L]$ and $\mathbb{V}[L]$ the mean and the variance of L, where λ_k is the p-norm of the radii of interest in the k-th state in the $list_d$ of Sect.IV-A, and w_1, w_2 are user-defined weigths. The variance is considered in this cost function so that the minimization of a radius at a given point does not lead to the growth of another radius at another waypoint. This function is 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. Therefore, it is unsuitable for global optimization using sampling-based motion planners like [21], [22], since they require additive and monotonic objective functions. Given that we do not have the analytic expression of the cost function derivatives, the accuracy optimization of A-Optim has to be performed by a derivative-free method. In this work, we simply used a robust version of the random shortcut algorithm [23] that performs robust collision checks (as in Alg. 2) to maintain the robustness of the initial trajectory computed by the R-SAMP planner.

V. APPLICATION TO A 3D-QUADROTOR

We first present the quadrotor dynamic model considered in this work and then further explain how uncertainty tubes were learned for this model.

A. System and controller

Let the ENU (East North Up) world frame be defined as $F_W = \{O_W, X_W, Y_W, Z_W\}$ and $F_B = \{O_B, X_B, Y_B, Z_B\}$ be the quadrotor body frame attached to its geometric center (O_B). The state of the quadrotor is defined as $\boldsymbol{q} = [\boldsymbol{x}\, \boldsymbol{v}\, \boldsymbol{\rho}\, \boldsymbol{\omega}]$ where $\boldsymbol{x} = [x\, y\, z] \in \mathbb{R}^3$ and $\boldsymbol{v} = [v_x\, v_y\, v_z] \in \mathbb{R}^3$ are respectively the position and velocity vector of O_B expressed in F_W. The body orientation w.r.t. F_W is represented by the unitary quaternion $\boldsymbol{\rho}$ and its angular velocity as $\boldsymbol{\omega} = [\omega_x\, \omega_y\, \omega_z] \in \mathbb{R}^3$. Finally, let $\boldsymbol{R(\rho)}$ be the rotation matrix associated to $\boldsymbol{\rho}$.

We consider that the center of mass is displaced from the robot's geometric center of an offset $\boldsymbol{x_c} = [x_{cx}, x_{cy}, x_{cz}]$ expressed in F_B. Under this consideration, the total force ($\boldsymbol{f_{tot}}$) and torque ($\boldsymbol{\tau_{tot}}$) acting on the quadrotor can be expressed in F_B s.t.

$$\boldsymbol{f_{tot}} = f Z_W - mg\boldsymbol{R(\rho)}^T Z_W - m[\boldsymbol{\omega}]_\times [\boldsymbol{\omega}]_\times \boldsymbol{x_c}$$
$$\boldsymbol{\tau_{tot}} = \boldsymbol{\tau} - mg[\boldsymbol{x_c}]_\times \boldsymbol{R(\rho)}^T Z_W - [\boldsymbol{\omega}]_\times (\boldsymbol{J} - [\boldsymbol{x_c}]_\times [\boldsymbol{x_c}]_\times)\boldsymbol{\omega}$$

where f and $\boldsymbol{\tau}$ are the propeller total thrust and torques, m is the mass and $\boldsymbol{J}$ is the inertia matrix of the system. By considering the spatial inertia matrix

$$\boldsymbol{S} = \begin{pmatrix} m\boldsymbol{I_3} & -m[\boldsymbol{x_c}]_\times \\ m[\boldsymbol{x_c}]_\times & \boldsymbol{J} - m[\boldsymbol{x_c}]_\times [\boldsymbol{x_c}]_\times \end{pmatrix}$$

one finally gets the body frame linear acceleration $\boldsymbol{\alpha}$ and angular acceleration $\boldsymbol{\eta}$ as: $\begin{pmatrix} \boldsymbol{\alpha}^T\, \boldsymbol{\eta}^T \end{pmatrix}^T = \boldsymbol{S}^{-1} \begin{pmatrix} \boldsymbol{f_{tot}}^T\, \boldsymbol{\tau_{tot}}^T \end{pmatrix}^T$. The dynamic model is then defined as follows:

$$\dot{\boldsymbol{q}} = \begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{v} \\ \dot{\boldsymbol{v}} = \boldsymbol{\alpha} \\ \dot{\boldsymbol{\rho}} = \frac{1}{2}\boldsymbol{\rho} \otimes \boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}} = \boldsymbol{\eta} \end{cases} \tag{6}$$

As tracking controller, we consider the so-called Lee (or geometric) controller [24] where the control inputs are the squared rotor speeds $\boldsymbol{u} = [\omega_1^2\, \omega_2^2\, \omega_3^2\, \omega_4^2]^T$ that are related to f and $\boldsymbol{\tau}$ by mean of a standard allocation matrix.

The uncertain parameters vector is defined as $\boldsymbol{p}$ = $[m,\, x_{cx},\, x_{cy},\, J_x,\, J_y,\, J_z]^T \in \mathbb{R}^6$, which represents parameters that are difficult to evaluate or likely to vary during execution. We chose as nominal parameters $\boldsymbol{p}_c$ = $[1.113, 0.0, 0.0, 0.015, 0.015, 0.007]^T$ and their associated uncertainty range $\delta\boldsymbol{p} = [7\%, 3cm, 3cm, 10\%, 10\%, 10\%]^T$, which represents the percentage variation of the parameters w.r.t. their associated nominal value, except for x_{cx} and x_{cy} whose nominal values are zeros.

B. Learning uncertainty tubes for the 3D-quadrotor

As mentioned in Sect. III-B, our GRU predicts the radii of the uncertainty tubes and the system inputs. In the quadrotor case, the control inputs to be predicted are $\boldsymbol{u} = [\omega_1^2\, \omega_2^2\, \omega_3^2\, \omega_4^2]^T$. Regarding the uncertainty tubes, the GRU predicts $\boldsymbol{r_q}$ =$[r_x,\, r_y,\, r_z]^T$ which represent the radii 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).

We choose the desired states as input parameters to the GRU, as they form the input to the control loop. The steering method used to plan these desired states is the *Kinosplines* of [25] and a desired state is defined as $\boldsymbol{q}_d = [\boldsymbol{\gamma}\, \dot{\boldsymbol{\gamma}}\, \ddot{\boldsymbol{\gamma}}]^T$, where $\boldsymbol{\gamma} = [x_d\, y_d\, z_d\, \Psi_d]^T$ with x_d, y_d, z_d the desired position along the x, y, z-axis respectively and Ψ_d the desired yaw angle.

In order to make our learning independent of workspace boundaries (i.e. position bounds and initial orientation), we choose to only use the desired linear and angular velocities and the accelerations to be the network input components. In other words, $\boldsymbol{q_{NN}^k}$ =$[\dot{x}_d\, \dot{y}_d\, \dot{z}_d\, \dot{\Psi}_d\, \ddot{x}_d\, \ddot{y}_d\, \ddot{z}_d]^T$ where k refers to the k-th state of the desired trajectory.

For the experiments presented below, the GRU was trained on 10,000 randomly generated trajectories (80% are used for the training set and 20% for the validation set) in an obstacle-free environment to be totally independent of the obstacles. Note that, since the learned GRU depends on the controller's nominal gains[2], the A-Optim method cannot benefit from it. More details regarding the dataset generation, training, and evaluation of the GRU can be found in [18].

VI. PERFORMANCE EVALUATION

This section first presents results showing the quality of the learning-based uncertainty tubes prediction and its high efficiency when used for robust motion planning. Then, we show the ability of the proposed R-SAMP planning approach to generate robust and accurate trajectories for the two experimental scenarios illustrated in Fig. 1.

A. Learning-based tube computation

In order to evaluate the accuracy of the GRU predictions, we generated a test set composed of 1,000 trajectories different to the ones used in the training and validation sets. The performance of the method is illustrated in Table I, which reports the MAE (Mean Absolute Error) of the norm of the output vector components $\boldsymbol{r_q}$, $\boldsymbol{u}$ and $\boldsymbol{r_u}$ computed on the validation and test sets. This high accuracy of the prediction is also illustrated in Fig. 3 that depicts the norm of predicted vector components for a 300-state trajectory of the test set w.r.t. their real values. It also highlights the high robustness of the model, especially during transient phases induced by velocity variations not seen during training but present in the test set. Predictions for each component independently are available in [18]. Finally, we mention that the predictions are only valid for the parameter maximum range $\delta\boldsymbol{p}$ chosen during the generation of the training set, and that the model is trained for given values of the controller gains.

Fig. 4 shows the significant performance improvement of using this learning-based prediction within a sampling-based tree planner for checking the robustness of the local tree expansions (see Alg. 2), against the previous version [10] that directly integrates the $\boldsymbol{\Pi}$ dynamics. Results provided for RRT and its RRT* near time-optimal variant compare the number of iterations of the main loop of the algorithm as a function of computing time in an obstacle-free environment,

[2]Learning optimal controller gains is left to future work, as it would require additional work on database generation and data annotation.

	Validation set			Test set		
	$\boldsymbol{r_q}$	$\boldsymbol{u}$	$\boldsymbol{r_u}$	$\boldsymbol{r_q}$	$\boldsymbol{u}$	$\boldsymbol{r_u}$
MAE	$2.6e^{-4}$	17.3	64.5	$1.1e^{-3}$	71.0	279.8

TABLE I: MAE (Mean Absolute Error) of the norm of the $\boldsymbol{r_q}$ $[m]$, $\boldsymbol{u}$ and $\boldsymbol{r_u}$ [(rad/s)²] components of the output vector computed on the validation and test sets.

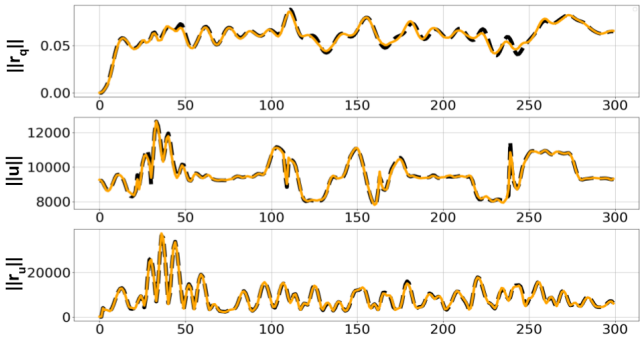
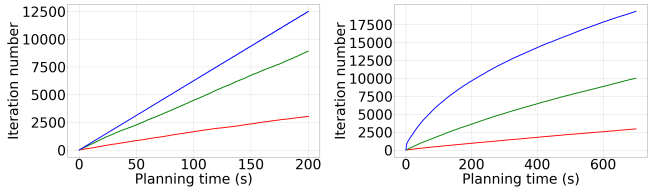


Fig. 3: Example of GRU predictions along a trajectory (orange) against true values (back). $||\boldsymbol{r_q}||, ||\boldsymbol{u}||$ and $||\boldsymbol{r_u}||$ refer to the norm of their respective vector components. $||\boldsymbol{r_q}||$ is expressed in m, and control input associated values ($||\boldsymbol{u}||, ||\boldsymbol{r_u}||$) are squared propeller speeds [(rad/s)²].



(a) RRT (b) RRT*

Fig. 4: Number of (a) RRT / (b) RRT* iterations as a function of planning time in an obstacle-free environment using the standard (non-robust) RRT/RRT* implementation (blue), compared to robust versions using the GRU-based tube prediction (green) or the integration of the dynamics of $\mathbf{\Pi}$ (red), as done in [10].

showing in both cases a significant time gain thanks to the proposed learning method compared with the method that integrates the dynamics of $\mathbf{\Pi}$. Note that in the case of RRT, this time gain is constant (3 times faster) because the expansion benefits from the neural network only once per iteration. In the case of RRT*, the denser the tree, the more robust collision tests are required for the rewiring connection phase. Therefore, much more time is saved when using the learning method. The gain on the planning time can reach more than one order of magnitude for problems requiring a significant amount of iterations.

B. Robust planning

We first demonstrate good efficiency and robustness of the R-SARRT planner (see Sect. IV-B) from comparative simulation results with a standard (non-robust) RRT, a RandUp-RRT [8] and SARRT standing for a RRT implementation of our former framework SAMP [10]. The robust collision checking of R-SARRT and SARRT is achieved by enlarging the robot shape to account for uncertainty. As for RandUp-RRT, it has been implemented with 20 "RandUP particles" to approximate the reachable set, and no ϵ-padding is used.

We also compared an asymptotically optimal version of our algorithm (R-SARRT*) to a classic RRT* to compute near time-optimal trajectories. Both algorithms ran until the solution cost converged below a threshold. Note that we cannot perform a comparison with the RandUp-RRT since there is no optimal version of the algorithm. The comparison is based on their planning time and success rate on the scenario depicted in Fig.1a, using an Intel i9 CPU@2.6GHz and a RTX A3000 GPU for the GRU predictions. The same geometric controller that steers the robot toward a sampled desired state $\boldsymbol{q}_d$ was used for all planners.

Table II shows comparative results averaged for each planner over 20 trajectories and 30 simulations with uncertain parameters in the range $\delta\boldsymbol{p}$ of Sect.V-B. First note that RandUp-RRT, SARRT and R-SARRT have a much stronger robustness than standard non-robust RRT. R-SARRT has a success rate of 100% compared to 99,2% for RanUp-RRT. Indeed, as mentioned in [8], [9], the computation of a conservative reachable set requires some additional padding step, which is set to zero in our experiment[3]. Also note the higher efficiency of R-SARRT which only uses one prediction per iteration whereas RandUp-RRT needs to perform a propagation per particle, yielding to a longer planning time. As for SARRT, it does not build a robust tree like R-SARRT. Instead, it only robustly checks the final solution, causing frequent disconnections and re-connections of non-robust nodes, which results in a higher planning time. RRT, which does not account for robustness, remains faster but with a significantly lower success rate. Similar results are observed on the optimal versions. An example of trajectories planned by RRT* and a robust version R-SARRT* optimizing the trajectory duration, and their associated simulations, is illustrated in Fig.5. It shows the effective robustness of the proposed algorithm as illustrated by the higher success rate indicated in Table II.

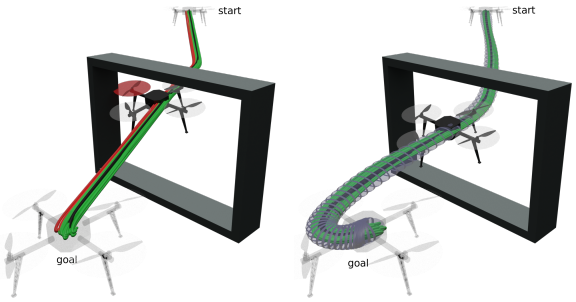
We also experimentally demonstrate the window scenario on a real quadrotor. Uncertainties are added to the system by randomly attaching a mass of up to 80g (not known by the controller) to the drone as depicted in Fig.6a.

In this experiment, a non-robust trajectory planned by RRT* and a robust one planned by R-SARRT* were executed ten times, using the same masses and attachment points between the two algorithms. All trajectories were planned offline on a remote computer. To make the robot execute them, the geometric controller [24] ran online on the quadrotor's onboard computer, tracking the trajectories provided as input. The robot state was measured using a motion capture system with millimeter accuracy, ensuring that the only source of uncertainty was the attached unknown mass. Fig.7 illustrates the experimental execution of a non-robust RRT* trajectory and a robust R-SARRT* trajectory. The figure shows the

[3]The padding value is a user parameter that is difficult to find. Choosing the wrong padding value can result in set estimations that are too conservative. We chose zero as in some experiments of [8].

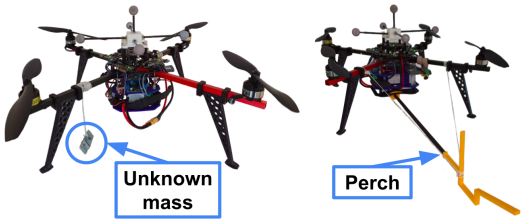
	Basic Planners				Asymptotically Optimal Planners	
	RRT	RandUp-RRT	SARRT	R-SARRT	RRT*	R-SARRT*
Success (%)	61.8	99.2	**100.0**	**100.0**	56.5	**100.0**
Plan time (s)	**7.1** ± 7.6	45.6 ± 33.4	57.8 ± 49.1	22.3 ± 15.8	**308.7** ± 235.7	584.3 ± 394.7

TABLE II: Average planning time and success rate (no crash) of the simulated motions planned by RRT, RandUP-RRT, our former planner SARRT and our R-SARRT, as well as RRT* and our R-SARRT* variants optimizing time, over 20 plans and 30 simulations per plan.



(a) RRT* (b) R-SARRT*

Fig. 5: Planned trajectory (black) produced by a (a) RRT* and our (b) R-SARRT*. Simulated trajectories under uncertainty are displayed in green in the case of success, and in red in the case of a crash.



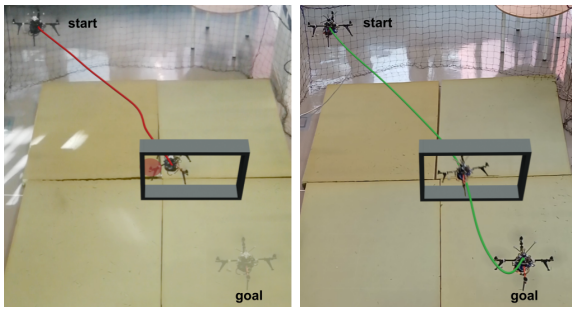
(a) (b)

Fig. 6: Quadrotor setups for the two scenarios considered for the experimental validation. (a) a drone equipped with a random mass to perform a robust navigation through a window (b) a drone equipped with a perch to catch the rings.

recorded executions within a virtual environment to detect virtual collisions, thus mitigating the risk of real crashes and damages to the robot. The experimental results confirm the simulation observations, providing an overall success rate of 100% in the case of the robust trajectory computed with R-SARRT*, against 40% for the classic RRT*.

C. Accuracy optimization

We implemented the A-Optim method of Sect.IV-C by using a robust version of the random shortcut algorithm [23] and (5) as cost function to be optimized, where the radii of interest are the ones along the x, ρ and ω components of q. At each iteration of this method, a shortcut is attempted between two states of the input trajectory that are randomly sampled together with the controller gain values, sampled between 50% and 150% of their nominal values. Fig.8 motivates why we optimized both the trajectory and the controller gains at the same time in the A-Optim function in order to minimize uncertainty for a given point, as mentioned



(a) RRT* (b) R-SARRT*

Fig. 7: Experimental execution by a quadrotor with uncertainty of trajectories planned by RRT* (a) and R-SARRT* (b). Both trajectories are executed with the same uncertainty and a virtual collision is found in the RRT* case while the R-SARRT* execution is robust.

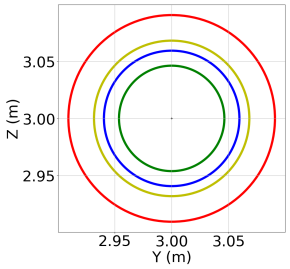


Fig. 8: Example of uncertainty ellipsoid without optimization (red), with local trajectory optimization (yellow), with gains optimization (blue), and with local trajectory and gains optimization at the same time (green).

in Sec.IV-A. In fact, these results corroborate the findings of [19], but this time by employing a sampling-based motion planner that considers obstacles in the environment.

We evaluated our complete framework with the accuracy optimization in a scenario that involves the in-flight retrieval of two 2cm radius rings in a cluttered environment using a drone equipped with a perch (see Fig.6b) in a (near) time optimal way. The experimental setup is shown in Fig.9. When the first ring is caught it becomes part of the drone and modifies the overall mass/inertia and center of mass of the system in an unmodeled way. A success is characterized by the recovery of both rings, otherwise we consider the execution as a failure.

We executed 10 trajectories using a vanilla (non-robust) RRT* planner and the RA-SARRT* algorithm, both of which optimize the trajectory time. The RRT* does not use the A-Optim method to optimize accuracy while the RA-SARRT* does, in addition to guaranteeing the robustness. The offline

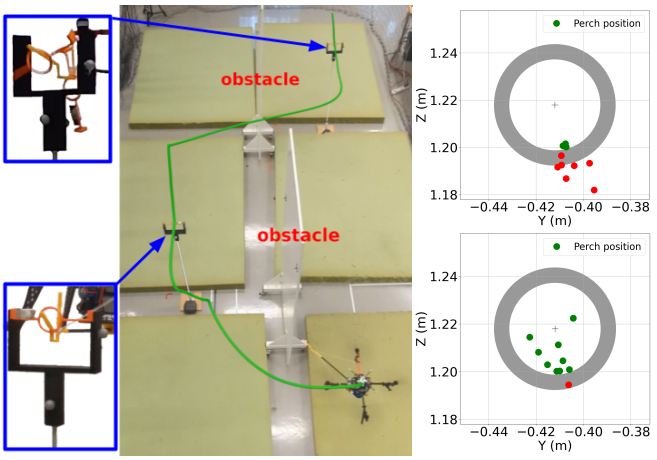


Fig. 9: Experimental validation of the "ring catching" scenario with a perch-equipped drone (left) with the position of the perch end-effector at the second ring location over 10 trajectories non accuracy optimized (top right) and accuracy optimized (bottom right).

optimization in A-Optim aimed at minimizing the uncertainty at the location of the two rings. Fig. 9 shows the perch end-effector position at the second ring location in the non-optimized case and in the optimized one. In the latter case, the perch tip is closer to the reference point in the middle of the ring than in the former case. This translates into a higher success rate of nine out of ten attempts to catch the ring with the optimized approach, against only three times out of ten for the non-optimized case. However, given the chosen system and controller parameters, there is no guarantee that the computed tube will be enclosed in within the ring. This explains why we still encounter one failure in the optimized case. Overall, the experimental results show a success rate of 90% for RA-SARRT* against only 30% for RRT*.

VII. CONCLUSION

We have presented a motion planner able to generate trajectories that are both robust and accurate in the presence of model uncertainties for a variety of robot/controller pair. The proposed planner leverages a GRU-based learning approach that quickly and accurately estimates the control inputs and the sensitivity-based uncertainty tubes of the state and of the inputs. The results on a quadrotor robot confirm the efficiency of the proposed learning method and highlight the benefit of its integration 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. An experimental demonstration involving a quadrotor UAV in a ring-catching task allowed to validate the approach in real conditions. Future works will focus on considering uncertainties not only in the dynamic model, by extending the computation of the tubes for state estimation uncertainties. Furthermore, we aim to expand the capabilities of the neural network to learn the optimal controller gains.

REFERENCES

[1] D. Limón, I. Alvarado, T. Alamo, and E. F. Camacho, "Robust tube-based mpc for tracking of constrained linear systems with additive disturbances," *Journal of Process Control*, vol. 20, no. 3, 2010.

[2] 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, 2010.

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

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

[5] 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, 2022.

[6] S. Singh, H. Tsukamoto, B. T. Lopez, S.-J. Chung, and J.-J. Slotine, "Safe motion planning with tubes and contraction metrics," in *IEEE CDC*, 2021.

[7] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," in *IEEE CDC*, 2017.

[8] A. Wu, T. Lew, K. Solovey, E. Schmerling, and M. Pavone, "Robust-rrt: Probabilistically-complete motion planning for uncertain nonlinear systems," in *The International Symposium of Robotics Research*. Springer, 2022.

[9] T. Lew, L. Janson, R. Bonalli, and M. Pavone, "A simple and efficient sampling-based algorithm for general reachability analysis," in *Learning for Dynamics and Control Conference*. PMLR, 2022.

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

[11] P. Brault, "Robust trajectory planning algorithms for robots with parametric uncertainties," Ph.D. dissertation, Université de Rennes, 2023. [Online]. Available: http://rainbow-doc.irisa.fr/pdf/2023_phd_brault.pdf

[12] P. Robuffo Giordano, Q. Delamare, and A. Franchi, "Trajectory generation for minimum closed-loop state sensitivity," in *IEEE ICRA*, 2018.

[13] P. Brault, Q. Delamare, and P. Robuffo Giordano, "Robust trajectory planning with parametric uncertainties," in *IEEE ICRA*, 2021.

[14] K. Cho, B. van Merriënboer, Ç. Gulçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 CEMNLP*, 2014.

[15] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *IEEE IROS*, 2018.

[16] R. S. Nair and P. Supriya, "Robotic path planning using recurrent neural networks," in *IEEE ICCCNT*, 2020.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[18] S. Wasiela, S. Ait Bouhsain, M. Cognetti, J. Cortés, and T. Simeon, "Note on learning sensitivity metrics for a quadrotor," July 2024, working paper or preprint. [Online]. Available: https://laas.hal.science/hal-04642304

[19] A. Srour, A. Franchi, and P. Robuffo Giordano, "Controller and trajectory optimization for a quadrotor uav with parametric uncertainty," in *IEEE IROS*, 2023.

[20] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

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

[22] 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, 2015.

[23] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *The international journal of robotics research*, vol. 26, no. 8, 2007.

[24] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *IEEE CDC*, 2010.

[25] A. Boeuf, J. Cortés, and T. Siméon, "Motion planning," *Aerial Robotic Manipulation: Research, Development and Applications*, 2019.