HAL
open science

A Sensitivity-Aware Motion Planner (SAMP) to Generate Intrinsically-Robust Trajectories

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

To cite this version:

Simon Wasiela, Paolo Robuffo Giordano, Juan Cortés, Thierry Simeon. A Sensitivity-Aware Motion Planner (SAMP) to Generate Intrinsically-Robust Trajectories. 2023 IEEE International Conference on Robotics and Automation (ICRA), May 2023, London, United Kingdom. hal-03808933v1

HAL Id: hal-03808933

https://laas.hal.science/hal-03808933v1

Submitted on 10 Oct 2022 (v1), last revised 6 Mar 2023 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Sensitivity-Aware Motion Planner (SAMP) to Generate Intrinsically-Robust Trajectories

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

***Abstract*— *Closed-loop state sensitivity* [1], [2] is a recently introduced notion that can be used to quantify deviations of the closed-loop trajectory of a robot/controller pair against variations of uncertain parameters in the robot model. While local optimization techniques are used in [1], [2] to generate reference trajectories minimizing a *sensitivity*-based cost, no global planning algorithm considering this metric to compute collision-free motions robust to parametric uncertainties has yet been proposed. The contribution of this paper is to propose a global control-aware motion planner for optimizing a state sensitivity metric and producing collision-free reference motions that are robust against parametric uncertainties for a large class of complex dynamical systems. Given the prohibitively high computational cost of directly minimizing the state sensitivity using asymptotically optimal sampling-based tree planners, the proposed RRT*-based SAMP planner uses an appropriate steering method to first compute a (near) time-optimal and kinodynamically feasible trajectory that is then locally deformed to improve robustness and decrease its *sensitivity* to uncertainties. The evaluation performed on planar/full-3D quadrotor UAV models shows that the SAMP method produces low sensitivity robust solutions with a much higher performance than a planner directly optimizing the sensitivity.**

I. INTRODUCTION

Modern motion planners can generate feasible and globally-optimal paths for high-dimensional systems and complex constraints/environments. However, most planners do not consider (and, even less, exploit) the unavoidable presence of the feedback controller that will track the planned trajectories of the robot, thus leading to poor intrinsic robustness and potential loss of optimality due to the controller actions at runtime.

The idea of merging planning with control for generating "robust planners" or more "global controllers" for dealing with the robustness problem in a more comprehensive way is however not completely novel in the robotics community. For example, the "feedback motion planning" approach introduced in [3] exploits the possibility of constructing explicit Lyapunov functions, but assuming a LQR controller and linearizing the robot dynamics around a nominal trajectory. This idea was extended in [4] by considering a polynomial form for the robot dynamics/control strategy. This method is however computationally heavy and it requires the off-line computation of a finite set of motion primitives with their associated "funnels" where the system state is guaranteed to evolve. Also, in the control-aware motion planner [5], an Inverse Kinematics controller is used as local planner to connect randomly sampled states within a RRT algorithm. Although this method relies on some coupling between planning and control, uncertainties/disturbances are not explicitly considered. Other robust planning techniques integrate a reasoning on uncertainty models. In particular *Belief*-planning is a popular alternative for providing a principled and general framework for dealing with uncertainties, especially using POMDPs [6] [7]. However, despite recent progress in efficient point-based solvers, planning in *belief*-space still suffers from computational bottlenecks that limit applicability to low dimensional problems.

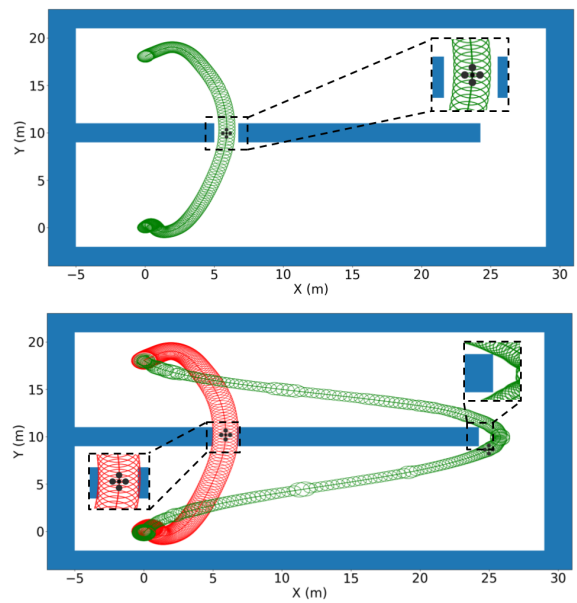

Fig. 1. Solutions with their uncertainty tube (in green) computed by our SAMP method between the same start/goal states of a quadrotor dynamical model for small/high parameters uncertainties (top/bottom, respectively)

Work has also been carried out on the control community side to propose "less local" controllers, resulting in the popular Model Predictive Control (MPC) technique [8] that iteratively replans an optimal (and feasible) trajectory with a feedback from the current robot state (that may deviate from the desired one because of disturbances/uncertainties). Robust versions of MPC-based schemes have also been investigated [9] for providing a robustness layer to the classical MPC formulation.

Recently, the notion of *closed-loop state sensitivity* has been introduced in [1], [2]. In these works, it is shown how to obtain the so-called *closed-loop state sensitivity*

* This work was supported by the project ANR-20-CE33-0003 "CAMP"
[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France, {swasiela, simeon, jcortes}@laas.fr
[2] CNRS, Univ Rennes, Inria, IRISA, Rennes, France, prg@irisa.fr

matrix $\mathbf{\Pi}$ which captures the sensitivity of the states of the robot/controller pair against variations in the model parameters. This sensitivity matrix $\mathbf{\Pi}$ is evaluated on the closed-loop system by explicitly considering any nonlinear systems dynamics and controllers. A metric based on $\mathbf{\Pi}$ (and on related quantities) has then been used for generating locally optimal trajectories under actuation and state constraints. However, this metric has never been used in a global planning framework, taking into account, for instance, presence of obstacles.

In this respect, this paper proposes a control-aware sampling-based planner call SAMP, that considers metrics introduced in [1], [2] in order to plan robust collision-free motions with a low cost w.r.t. the *sensitivity*-based metrics. This approach differs from [4] by considering general forms of controllers and systems, allowing also to get rid of the pre-computation and thus not to be limited to a finite set of trajectories. In addition, the proposed approach allows to also take into account uncertainties in the robot input, which may be relevant when considering actuation constraints.

The rest of this paper is organized as follows. Section II recalls the main notions of the considered *closed-lood sensitivity* metrics. Section III then first discusses the difficulties associated with using the sensitivity in classical global planners, and then presents our SAMP method. Simulations in different environments taking into account the full dynamics of a 3D quadrotor are provided in Sect. IV. Finally we draw some conclusions and future directions in Sect. V.

II. SENSITIVITY-BASED METRIC

In this section, we briefly recall the notion of *closed-loop state sensitivity* introduced in [1], [2] and later extended in [10], and discuss how this quantity can be used as a suitable metric for our planner.

We consider a generic robot dynamical model

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

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) parameters of the robot model. We also consider a reference trajectory $\boldsymbol{r}_d(\boldsymbol{a},\, t)$ parameterized by a vector $\boldsymbol{a}$ (e.g., the control points of a B-spline) and time. The reference trajectory $\boldsymbol{r}_d(\boldsymbol{a},\, t)$ is tracked by the robot by means of a controller in the generic form

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

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. Vector $\boldsymbol{p}_c$ may differ from the real $\boldsymbol{p}$ due to potential uncertainties in the knowledge of some robot parameters.

As discussed in [1], [2], it is possible to quantify how variations in $\boldsymbol{p}$ (w.r.t. the 'nominal' $\boldsymbol{p}_c$) will affect the states $\boldsymbol{q}(t)$ and inputs $\boldsymbol{u}(t)$ behavior for the closed-loop system (1–2) by means of the *state sensitivity matrix*

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

and *input sensitivity matrix*

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

These two matrices can be easily computed (numerically by forward integration) for the robot/controller pair (1–2) and they can be used for optimization purposes. For instance, one may try to find a reference trajectory $\boldsymbol{r}_d(\boldsymbol{a},\, t)$ that minimizes a norm of $\mathbf{\Pi}(t)$ (and/or of $\mathbf{\Theta}(t)$) for producing a 'motion plan' that would be intrinsically robust against uncertainties in the robot parameters $\boldsymbol{p}$.

A recent extension [10] has also shown how to leverage the quantities $\mathbf{\Pi}$ and $\mathbf{\Theta}$ for evaluating the tubes of perturbed state/input trajectories given a range for the parametric deviations. In particular, assume that each uncertain parameter p_i lies in the range $p_i \in [p_{c_i} - \delta p_i, p_{c_i} + \delta p_i]$ and consider the diagonal weight matrix $\boldsymbol{W} = \mathrm{diag}(\delta p_i^2)$. Letting $\Delta\boldsymbol{p} = \boldsymbol{p} - \boldsymbol{p}_c$, an ellipsoid in the parameter space centered at $\boldsymbol{p}_c$ and with semi-axes δp_i has equation

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

from which, as explained in [10], it is possible to obtain the resulting ellipsoids in the state space

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

and in the input space

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

Here $\Delta\boldsymbol{q}$ stands for $\Delta\boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_{nom}$ where $\boldsymbol{q}_{nom}$ is the state behavior in the nominal case ($\boldsymbol{p} = \boldsymbol{p}_c$), and analogously for $\Delta\boldsymbol{u}$.

Equation (6) can be used to define a sensitivity metric that can capture the deviations in the states given the (known) parametric ranges δp_i. To this end, as discussed in [10], in this work we consider the largest eigenvalue of the kernel matrix $\mathbf{\Pi}(t)\boldsymbol{W}\mathbf{\Pi}(t)^T$ as *sensitivity norm*, since this will represent the largest (worst-case) deviation in the state space. In particular, letting $\lambda_i(t)$ be the eigenvalues of $\mathbf{\Pi}(t)\boldsymbol{W}\mathbf{\Pi}(t)^T$, we approximate the $\max(\cdot)$ operation with the p-norm

$$\lambda_{max}(t) \approx \left(\sum \lambda_i(t)^p\right)^{1/p} \tag{8}$$

for a large enough p. The total cost function $c[\pi]$ for a trajectory π is then defined as

$$c[\pi] = \int_{t_0}^{t_f} \lambda_{max}(\tau) d\tau. \tag{9}$$

Minimization of this cost will minimize the largest deviation of the state tube along the whole motion.

A second possible use of (6–7) is to evaluate the radius $r_i(t) \geq 0$ of the perturbed tubes along directions of interest in the state and/or input spaces, see [10] for a detailed

derivation. The quantities $r_i(t)$ can then be used to bound components of the states $\boldsymbol{q}(t)$ or inputs $\boldsymbol{u}(t)$ over time. For example, letting $q_{i,nom}(t)$ the behavior of the i-th state component in the non-perturbed case ($\boldsymbol{p} = \boldsymbol{p}_c$), one has that

$$q_{i,nom}(t) - r_i(t) \leq q_i(t) \leq q_{i,nom}(t) + r_i(t) \tag{10}$$

for a perturbed state $q_i(t)$, and an analogous derivation is also possible for the inputs $u_i(t)$. Availability of bounds 10 can be exploited for, e.g., planning a trajectory that will ensure *robust* collision avoidance for the envelope of perturbed states $\boldsymbol{q}$ for any value of the parameters $\boldsymbol{p}$ in the considered range.no

III. SENSITIVITY-AWARE MOTION PLANNING

We will now present our SAMP approach that consists of two stages: it first plans a (near) time-optimal, collision-free and kinodynamically feasible trajectory that is then locally optimized for minimizing the cost function (9) via a 'shortcut' technique classically used to smooth solutions of randomized planners [11]. In the first part of this section, we will justify the choice of this approach and in the second part we will detail our implementation.

A. General approach

1) Sensitivity Computation: As explained in Sec.II, the state/input sensitivity matrices $\boldsymbol{\Pi}(t)$ and $\boldsymbol{\Theta}(t)$ are computed by forward integration. This computation is performed along the reference trajectory using a given time step. Thus, depending of the execution time of the trajectory, the evaluation of the sensitivity (and computation of the associated uncertainty tube) can be time consuming, in the order of hundred milliseconds for a hundred points over the trajectory, and up to several seconds for long trajectories requiring thousands of evaluations.

Also note that the evaluation of the sensitivity is done along a reference trajectory, taking into account both the dynamics of the system and the controller. Therefore, in cases where this trajectory is not feasible with respect to the system dynamics, there is a risk that despite the presence of a robust controller, the tracking error will diverge, making the sensitivity calculation impossible. Thus, this justifies the requirement for the SAMP method of computing kinodynamically feasible trajectories in order to guarantee successful *sensitivity* computations.

2) Sensitivity-Aware Motion Planner (SAMP): One may consider to use standard asymptotically-optimal random-tree planners like RRT* [12] or SST* [13] for computing robust motions directly minimizing the sensitivity metrics. However such solution would be too inefficient for the reasons explained below.

Regarding RRT*, each iteration is based on an extension phase towards a random sample followed by a rewiring phase, both considering a neighborhood of the sampled state. Since this neighborhood consists of a growing set of possibly many vertices, and as the two phases mentioned above require to compute the sensitivity for each feasible trajectory among this set, this leads to a prohibitive computation time per iteration.

Moreover, it is important to emphasise that the sensitivity is not invariant for non-modified edges since $\boldsymbol{\Pi}$ depends on the path taken from the root node to the source node of the edge. Therefore, unlike a distance or time cost which are invariant on non-modified edges, the sensitivity must be recomputed for an edge each time one of its parents is changed. Thus, when the rewiring is performed and the cost of the rewired nodes is updated, the sensitivity must also be recalculated on the edges. This re-computation can lead to sub-optimal connections and therefore a re-connection is necessary.

Consequently, the computation time of a RRT* iteration becomes rapidly too high (up to tens of seconds), leading to a very large running time with a slow convergence of the algorithm as shown in Fig.2. In order to improve the speed of random tree planners one needs to limit the number of sensitivity path-costs computed per iteration. This is actually the case for the SST* algorithm that indeed results in a much higher performance than RRT* as shown in Fig.2. However, the convergence time of SST* still remains very long because in the absence of a steering method many iterations must be performed in order to reach a goal region accurately. We therefore propose a new approach called SAMP (Sensitivity-Aware Motion Planner) described in Algorithm 1. SAMP is aimed at producing an intrinsically-robust and control-aware motion with a good *sensitivity*-cost in a much faster manner than the conventional planners mentioned above. It consists in planning a (near) time-optimal, kinodynamically feasible and uncertainty-robust trajectory using a modified version of RRT* (SARRT*) to guarantee this robustness and then performing a local optimization of the sensitivity of the computed solution. This second step is currently performed using a random 'shortcut' algorithm [11].

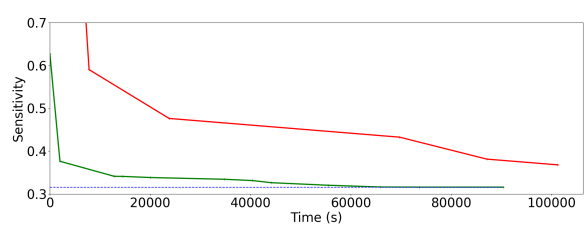

Fig. 2. Evolution of the sensitivity as a function of the running time of an SST* (in green), and of an RRT*(in red)

Algorithm 1 SAMP $[x_{init}, x_{goal}]$

1: $sol_{SARRT^*} \leftarrow SARRT^*(x_{init}, x_{goal})$;
2: $sol_{\Pi} \leftarrow SAshortcut(sol_{SARRT^*})$;
3: **return** sol_{Π}

B. Description of SAMP

1) Steering Method: Since the cost function of eq. (9) corresponds to the integration of the sensitivity over the execution time of the trajectory, it is then noticeable that

starting from an initial (near) time-optimal trajectory helps to produce initial solutions of rather good quality wrt. the sensitivity cost. In order to produce a near time-optimal and kinodynamically feasible trajectory, the SARRT* planner currently uses as steering method the *Kinosplines* defined in [14]. To ensure kinodynamic feasibility, this method considers the kinodynamic states of the system and two given states are connected using a bang-bang snap control. Regarding time optimality, the local optimization used to steer between the two states aims to reach full speed as quickly as possible and to maintain it as long as possible. This steering method, as well as the sensitivity-based metric, is applicable to a large class of dynamic systems, which makes the proposed framework very general. Note that, instead of using the cost-to-go as distance metric, an efficient state-space quasi-metric is employed as defined in [14].

2) Sensitivity-Aware RRT (SARRT*):* Algorithm 2 provides the pseudo-code of the SARRT* planner. It is a variant of RRT*, modified in order to guarantee the robustness of the computed solution wrt. the state/input uncertainties. Indeed, in addition to providing a kinodynamic (near) time-optimal trajectory the algorithm verifies that the returned trajectory is collision-free wrt. the state uncertainties and also that inputs remain in their allowed bounds. However, as mentioned in Sec.III-A.2, the number of sensitivity computation within the RRT* must remain as limited as possible to avoid a too long computing time. For this reason, trajectory robustness is checked in a lazy way, only when a better solution is found, and reconnecting the nodes optimally if necessary.

The first stage applies the unmodified RRT* (line 2-5) where the *Extend* procedure performs the collision checking, optimal connection, and rewiring phase for the given sample x_{rand} as in [12]. Then, the algorithm checks if a new solution with a better cost has been found (line 6). If so, it computes the uncertainties along the the trajectory (line 7-8).

Next, function *TubesCC* (line 9) performs a robust Collision Checking by first considering the state uncertainties (computed using eq.(6)) and then 'growing' the geometry of the robot consequently. It also considers the uncertainties on the different actuators by checking that the tube associated with each input remains in its feasibility domain. An example of infeasible input is presented in Fig.3, where the tube (red) around the reference trajectory (blue) exceeds the maximum allowed input (red). This *TubeCC* function checks if the motion is valid or not considering the uncertainties. In case of collision, it also determines the first colliding tree node (state) along the path. This node is used to perform an optimal re-connection of the tree starting from it.

This re-connection is performed by the *RobustExtend* function (line 13) that follows the same principle as the *Extend* fucntion in the standard RRT*, excepted that it only considers for the optimal connection and rewiring stages the neighborhoods of nodes reached by a robust trajectory wrt. the state/input uncertainties. For this, it maintains a set denoted $X_{collide}$ of all states already found by *TubeCC* to be invalid wrt. the uncertainty tubes during the previous iterations. This set is unique for each node of the tree. It is

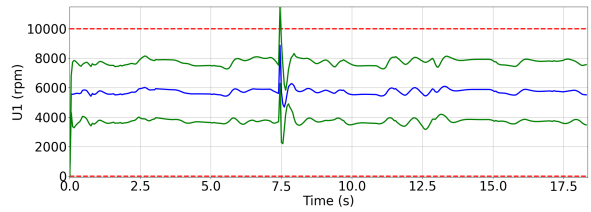

Fig. 3. Profile of the nominal input 1 (in blue) for a quadrotor controlled in rotation per minute (rpm), and its uncertainty tube (in green), along the trajectory considering the maximum and minimum allowed inputs (in red)

first incremented for $x_{collide}$ by its current parent if $x_{collide}$ is eventually found at the current iteration. Then, all the states of the tree from $x_{collide}$ are disconnected, and for each of them an *Extend* procedure is executed considering the associated $X_{collide}$ set. As in RRT*, an optimal re-connection is first tried. If no re-connection is found for a node, then the node is deleted from the tree. Otherwise, the rewiring phase is carried out considering the $X_{collide}$ set.

The output of SARRT* is a (near) time-optimal trajectory, feasible both in terms of kinodynamic constraints and also with respect to uncertainties. It is however not optimised in terms of the sensitivity. Therefore, its robustness can be further improved.

Algorithm 2 SARRT*$[x_{init}, x_{goal}]$

1: $V \leftarrow \{x_{init}, x_{goal}\}$; $E \leftarrow \emptyset$; $sol \leftarrow \emptyset$; $i \leftarrow 0$;
2: **while** $i < N$ **do**
3: $T \leftarrow (V, E)$;
4: $x_{rand} \leftarrow Sample(i); i \leftarrow i + 1$;
5: $T \leftarrow Extend(T, x_{rand})$;
6: $sol_{new} \leftarrow CheckForSolution(T)$;
7: **if** sol_{new} $found$ **then**
8: $U \leftarrow GetUncertainties(sol_{new})$;
9: $\{valid, x_{collide}\} \leftarrow TubesCC(sol_{new}, U)$;
10: **if** $valid$ **then**
11: $sol \leftarrow sol_{new}$;
12: **else**
13: $T \leftarrow RobustExtend(T, x_{collide})$;
14: **end if**
15: **end if**
16: **end while**
17: **return** sol

3) Sensitivity-Aware shortcut (SAshortcut): In order to improve the sensitivity-based cost function, a local optimization is performed using a simple variant of the 'shortcut' smoothing algorithm [15], called SAshortcut and described in Algorithm 3. The reason for this variant, as already mentioned in Sec.III-A.2, is that the sensitivity of a trajectory portion is not invariant but instead depends on the previous portions since the start of the motion.

The algorithm first randomly samples two states along the trajectory (line 3). Then, it computes the Kinospline between the two samples (line 6) and verifies that it is collision-free (line 7). Unlike the classical shortcut, where only the cost

of the old and the new portions are compared, the whole trajectory must be reconsidered in order to incrementally compute the sensitivity and the related uncertainty "U" using the $GetSensi$ procedure (line 8-9). Finally, if the new solution has a lower sensitivity cost (line 10), then a robust collision checking using the uncertainty tubes is performed before updating the trajectory portion in case of success (line 11-12).

Algorithm 3 SAshortcut $[sol_{SARRT^*}]$

1: $\{sol_\Pi, cost_\Pi\} \leftarrow \{sol_{SARRT^*}, GetSensi(sol_{SARRT^*})\}$;
2: **while** $i < N$ **do**
3: $\quad \{x_1, x_2\} \leftarrow SampleOnTraj(sol_\Pi)$;
4: $\quad \pi_{start} \leftarrow sol_\Pi(x_{init}, x_1)$;
5: $\quad \pi_{end} \leftarrow sol_\Pi(x_2, x_{goal})$;
6: $\quad \pi \leftarrow Steer(x_1, x_2)$;
7: $\quad$ **if** $CollisionFree(\pi)$ **then**
8: $\quad\quad sol_{new} \leftarrow \pi_{start} + \pi + \pi_{end}$;
9: $\quad\quad \{cost_{new}, U\} \leftarrow GetSensi(sol_{new})$;
10: $\quad\quad$ **if** $CostBetterThan(cost_{new}, cost_\Pi)$ **then**
11: $\quad\quad\quad$ **if** $TubesCC(sol_{new}, U)$ **then**
12: $\quad\quad\quad\quad \{sol_\Pi, cost_\Pi\} \leftarrow \{sol_{new}, cost_{new}\}$;
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end if**
15: $\quad$ **end if**
16: $\quad i \leftarrow i + 1$;
17: **end while**
18: **return** sol_Π

IV. SIMULATION RESULTS

To evaluate the effectiveness of our approach, we considered the model of a 3D quadrotor as described in [15]. The control inputs are the squared rotor speeds of the quadrotor $\boldsymbol{u} = [\omega_1^2\, \omega_2^2\, \omega_3^2\, \omega_4^2]$. We consider $\boldsymbol{p} = [k_f\, k_\tau\, g_x\, g_y]$ as the uncertain parameters of our model where k_f and k_τ are coefficients associated with the dynamics of the propellers,

	U-shape	2-Way$_{low}$	2-Way$_{high}$	3D
Time SST* (s)	77529	123347	129765	178461
Time SARRT* (s)	**695** ± 225	**2841** ± 187	**3597** ± 233	**2678** ± 183
Time SAshortcut (s)	**553** ± 184	**390** ± 190	**849** ± 91	**607** ± 227
SAMP time gain (%)	98,4	97,4	96,6	98.3
Cost SST*	0.317	0.292	5.127	0.510
Cost SAMP	**0.325** ± 0.002	**0.297** ± 0.003	**5,283** ± 0.073	**0.519** ± 0.018
Sub-optimality of SAMP (%)	2,37	1,58	3,03	1,89

TABLE I

AVERAGE VALUES OF COSTS AND COMPUTING TIMES FOR THE DIFFERENT METHODS IN SEVERAL ENVIRONMENTS. STANDARD DEVIATIONS ARE PROVIDED FOR THE SAMP RESULTS ONLY AS THE SST* RESULTS DO NOT CONTAIN ENOUGH RUNS.

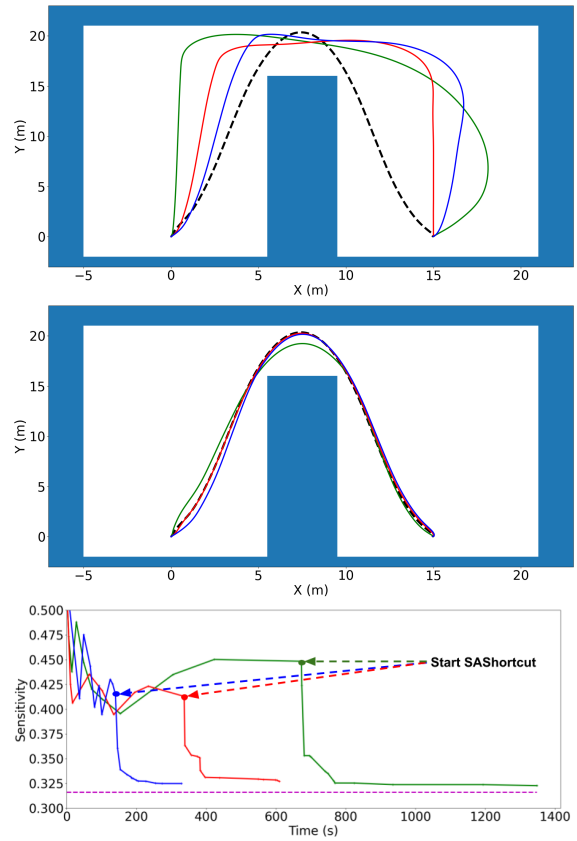

Fig. 4. Three trajectories (red, green, blue) produced by SARRT* (top), and the three trajectories resulting from SAMP (middle), with the evolution of their respective sensitivity (bottom). The dashed black trajectory (top, middle) and the dashed purple cost (bottom) correspond to the *sensitivity*-optimal reference found by SST*.

and g_x and g_y are the location of the centers of mass in the x-axis and y-axis of the quadrotor body frame, which may be uncertain due to the presence of on-board sensors for example. The values of this vector considered in the simulations are either $\delta\boldsymbol{p}_{low} = [10\%, 10\%, 5cm, 5cm]$ or $\delta\boldsymbol{p}_{high} = [25\%\ 25\%\ 10cm\ 10cm]$, where the first two components are a percentage of their associated nominal values. Note that $\delta\boldsymbol{p}_{high}$ represents the maximum bounds that can be considered for our pair system/controller. Finally, the tracking controller considered in this work is the so-called Lee (or geometric) controller [16].

Let $\boldsymbol{X}_C = [x\, y\, z\, \theta\, \phi\, \psi] \in \mathbb{R}^6$ be the configuration of the quadrotor. We therefore define $\boldsymbol{X}_K = [\boldsymbol{X}_C\, \dot{\boldsymbol{X}}_C\, \ddot{\boldsymbol{X}}_C] \in \mathbb{R}^{18}$ as the kinodynamic state considered for motion planning.

The performances of the SAMP algorithm were evaluated in three different environments, also considering different parametric uncertainties. For the 2D environments (U-shape and 2-Way) we forced the drone to evolve on a plane by producing Kinosplines only between samples in it and by not allowing displacements along Z. The sensitivity-optimal trajectories used as references to evaluate SAMP were computed using an SST* as implemented in OMPL [17], by specifying a goal state $\boldsymbol{X}_{goal}$, and allowing a tight goal region around it for all the components of $\boldsymbol{X}_K$.

Table I gathers the average computing times of SST* and of the two core procedures of SAMP (SARRT*, SAshortcut), as well as the average final cost found by SST* and SAMP. The mean values associated with SST* were obtained over 3 runs (due to the very high computing time) while those associated with SAMP are averaged over 10 runs.

1) 2D U-Shape Environment: Fig.4 shows how SAMP is able to produce solutions that mimic the optimal one found by the SST* in the so called U-shape environment considering the $\delta \boldsymbol{p}_{low}$ parametric uncertainty vector. Furthermore, according to the results shown in Table.I and in Fig.4, SAMP produces solutions with sensitivity costs close to the optimal reference found by SST*. Also note the huge time saving of the SAMP method compared to the SST*.

Finally, we note that the average computing time of SARRT* in this case is low compared to the other environments. This is mainly due to the fact that in this case the time-optimal trajectory is far from the obstacles and therefore few re-connections via the SARRT*'s *RobustExtend* method are performed when extending the tree.

2) 2D 2-way Environments: This environment referred to Fig.1 where the two sets of uncertainties $\delta \boldsymbol{p}_{low}$ and $\delta \boldsymbol{p}_{high}$ are considered.

As shown in Fig.1, in the presence of small uncertainties SAMP produces a trajectory allowing to use the narrow passage. However, while considering large uncertainties, the time optimal trajectory which goes through the narrow passage cannot be taken as a collision is found by considering the uncertainty tube as shown in red in the Fig.1. Nevertheless, SAMP is able to find the fastest trajectory apart from those passing through this passage.

In both cases the average cost of the solutions produced by SAMP is close to that found by SST* as shown in Table I, where 2-Way$_{low}$ refers to the presence of small uncertainties and 2-Way$_{high}$ to the presence of large uncertainties. The average computing time of the SST* is equivalent in both cases as it is performed on exactly the same environment. However, note that the average computing time of the SARRT* is longer than in the U-shape case because the time-optimal trajectories must pass through a location where many collisions may occur. Moreover, in the 2-Way$_{high}$ case the SARRT*'s computing time is longer than in the 2-Way$_{low}$. This is because more iterations are needed before considering a neighborhood deprived of nodes passing through the narrow corridor. Finally, we see that the SAshortcut computing time remains of the same order of magnitude for both cases as that of the U-shape, and again SAMP produces a solution much faster than SST*.

3) 3D Environment: The uncertainty set considered in this full 3D environment is $\delta \boldsymbol{p}_{low}$. As shown in Fig.5, once again SAMP produces trajectories that mimic the optimal obtained by the SST*. It is interesting to note the impact of the SAshortcut procedure on the Z component in addition to the impact on the X Y components already seen in the previous 2D environments. The average computing time of the shortcut is similar to other environments according to Table I. Again, the SAMP method produces a near-optimal trajectory much faster than conventional optimal planners.

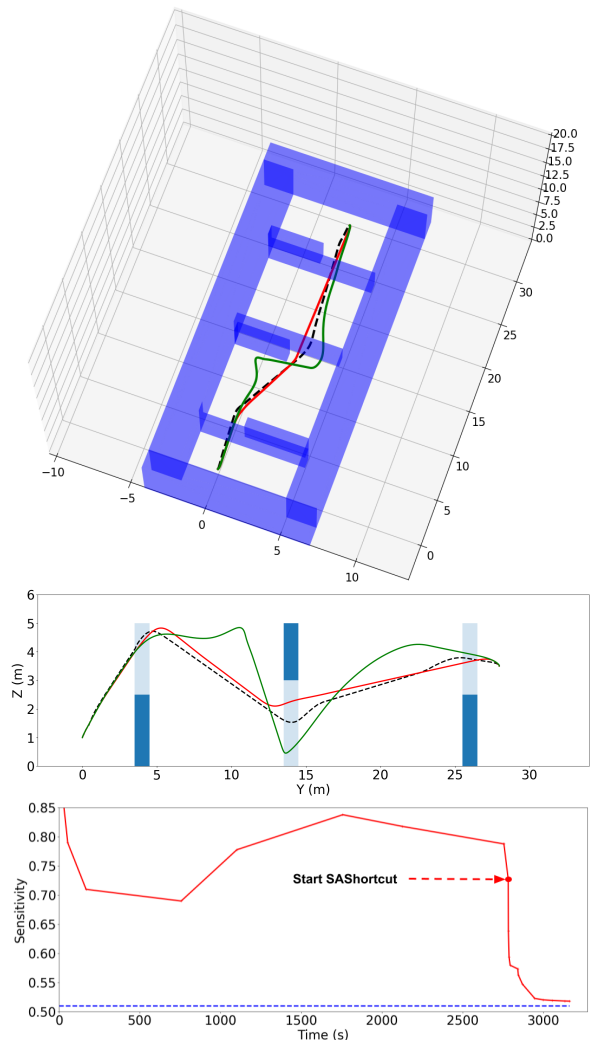

Fig. 5. Trajectories produced by SARRT* (green), SAMP (red), and SST* (dashed black) in a 3D environment (top). A profile view along the YZ plane is given (middle) as well as the evolution of the sensitivity (red) and the optimal one found by the SST* (dashed blue) (bottom) during the execution of SAMP.

V. CONCLUSION

We have presented a method called SAMP for planning control-aware trajectories robust to parametric uncertainties. The proposed framework has been specifically designed to handle global planning using *sensitivity*-based metrics from [1], [2], [10] in a more efficient way than the classical asymptotically optimal sampling-based tree planners. Simulations have shown that SAMP generates near-optimal trajectories with significant time savings, and which can be robustly executed by the considered robot for any value of the uncertain parameters in a given range. However, this computing time is still far from being applicable to real-time motion planning. Future work will therefore focus on improving computing time by considering learning techniques in order to estimate the sensitivity cost and uncertainties associated with a trajectory.

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] 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.

[4] 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.

[5] 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.

[6] S. Koenig and R. Simmons, "A robot navigation architecture based on partially observable markov decision process models," *Kortenkamp Al-ROBOTS*, 1998.

[7] S. Candido and S. Hutchinson, "Minimum uncertainty robot path planning using a pomdp approach," in *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2010, pp. 1408–1413.

[8] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

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

[10] P. Brault and P. Robuffo Giordano, "Temp," submitted to the 2023 Int. Conf. on Robotics and Automation, http://rainbow-doc.irisa.fr/pdf/ICRA_2022_CLS.pdf, 2023.

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

[12] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 7681–7687.

[13] 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.

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

[15] C. Bohm, P. Brault, Q. Delamare, P. Robuffo Giordano, and S. Weiss, "Cop: Control & observability-aware planning," in *2022 IEEE Int. Conf. on Robotics and Automation*, 2022.

[16] 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.

[17] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.