Note on learning sensitivity metrics for a quadrotor

Simon Wasiela, Smail Ait Bouhsain, Marco Cognetti, Juan Cortés, Thierry Simeon

Note on learning sensitivity metrics for a quadrotor

Simon Wasiela[1], Smail Ait Bouhsain[1], Marco Cognetti[1], Juan Cortés[1], and Thierry Siméon[1]

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

Abstract

Robust trajectory planning can involve the use of 'uncertainty tubes' which bound the system's states and inputs. These tubes can be derived from a variety of metrics, involving complex simulation computations. This is the case for the sensitivity-based uncertainty tubes computation we consider in this work. In fact, this computation was shown to be computationally expensive, and even more in the context of sampling-based planners, where it has to be performed at least once for each new sample. In order to solve this problem, a learning-based approach is presented in this work to predict these sensitivity-based uncertainty tubes. The report is organized as follows: first, it recalls the sensitivity basis and the tube computation, then presents the chosen system and controller. Next, the network architecture is presented, and finally the results including dataset generation, training to different types of network, from LSTMs and RNNs to GRUs, and evaluation are given in a final section.

1 Reminder: Closed-loop sensitivity

The notion of *closed-loop sensitivity* was introduced in [1, 2] 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.,

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

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 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 [1] and [2], 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 [1, 2] 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 the radius $r_{q,i}(t)$ related to $\boldsymbol{\Pi}(t)$ by mean of the following equation:

$$r_{q,i}(t) = \sqrt{\boldsymbol{n_i}^T \boldsymbol{K_\Pi(t)} \boldsymbol{n_i}}. \tag{5}$$

where $\boldsymbol{n_i} \in R^n$ is the i-th direction we are interested in and $\boldsymbol{K_\Pi}(t) = \boldsymbol{\Pi}(t)\boldsymbol{W}\boldsymbol{\Pi}(t)^T$ with $\boldsymbol{W}$ a diagonal weight matrix using δp (see [3] for details). Note that such bounds apply to the *nominal state* $(\boldsymbol{q_n})$[1] and that similar tubes can be obtained for the inputs.

Such radii computation relies on the knowledge of $\boldsymbol{\Pi}(t)$ and $\boldsymbol{\Theta}(t)$ which are computed numerically by integrating their respective dynamics (3). Depending on the number of parameters and on the dimension of the system state, this integration may have a non-negligible computational time. For example, referring to our quadrotor case in Sect.2, this requires solving a hundred ordinary differential equations. For trajectories composed of a hundred samples, this may require from tens to hundreds of milliseconds.

2 Quadrotor dynamics and control

Fig.1 represent the quadrotor state space where the ENU (East North Up) world frame is 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

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

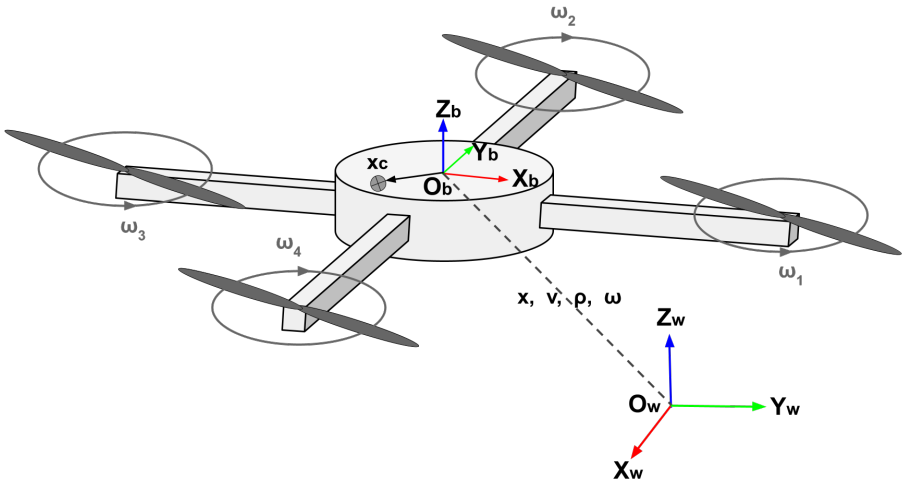

Figure 1: The quadrotor space with a shift in the center of mass.

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.

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

where f and $\boldsymbol{\tau}$ are the propeller total thrust and torques, m is the system mass and $\boldsymbol{J}$ is the inertia matrix. 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: $\left(\boldsymbol{\alpha}^T\ \boldsymbol{\eta}^T\right)^T = \boldsymbol{S}^{-1} \left(\boldsymbol{f_{tot}}^T\ \boldsymbol{\tau_{tot}}^T\right)^T$. 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 [4] where the control inputs are the squared rotor speeds $\boldsymbol{u} = [\omega_1^2\, \omega_2^2\, \omega_3^2\, \omega_4^2]^T$.

This controller first computes as follow the desired thrust f_d and torques $\boldsymbol{\tau_d}$ to be able to track the desired trajectory:

$$\begin{aligned} f_d &= (-\boldsymbol{k_x e_x} - \boldsymbol{k_v e_v} + mg\boldsymbol{e_3} + m\boldsymbol{a_d})\boldsymbol{R e_3} \\ \boldsymbol{\tau_d} &= -\boldsymbol{k_R e_R} - \boldsymbol{k_\omega e_\omega} \end{aligned} \tag{7}$$

where $\boldsymbol{e_x}$ and $\boldsymbol{e_v}$ respectively represent the error vector according to the desired positions and velocities, $\boldsymbol{a_d}$ represents the desired acceleration vector and $\boldsymbol{e_3} = [0\,0\,1]^T$. Then, the squared rotor speed are computed by mean of an allocation matrix $\boldsymbol{G}$ evaluated at the estimated nominal parameters $\boldsymbol{p}_c$ s.t.

$$\boldsymbol{u} = \boldsymbol{G(p_c)^{-1}} \begin{bmatrix} f_d \\ \boldsymbol{\tau_d} \end{bmatrix}$$

Finally, the computed control inputs are related to f and $\boldsymbol{\tau}$ by mean of an allocation matrix $\boldsymbol{G}$ evaluated at the (potentially unknown) system parameters $\boldsymbol{p}$ s.t.

$$\begin{bmatrix} f \\ \boldsymbol{\tau} \end{bmatrix} = \boldsymbol{G(p)u}$$

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 null. The nominal controller gains used are $\boldsymbol{k_x} = [20.0\,, 20.0\,, 25.0]$, $\boldsymbol{k_v} = [9.0\,, 9.0\,, 12.0]$, $\boldsymbol{k_R} = [4.6\,, 4.6\,, 0.8]$, $\boldsymbol{k_\omega} = [0.5\,, 0.5\,, 0.08]$.

3 Neural network architecture

A simplified representation of our neural network architecture applicable to RNN, GRU or LSTM is presented in Fig. 2. Blue blocks refer to the input of the neural network and are composed of a sequence of states, denoted by $\boldsymbol{q}_{NN}^k$, evaluated at the k-th time step of a desired trajectory, and an initial hidden state $hidden_0$.

The output of the neural network correspond to the green blocks. It is composed of a sequence of the predicted nominal control input values $\boldsymbol{u}^k$, and of $\boldsymbol{r_q}^k$ and $\boldsymbol{r_u}^k$, which represent the predicted radii of the uncertainty tubes along the desired direction of the state and of the input spaces, respectively. Finally, $hidden_F$ corresponds to the hidden state at the last point of our sequence (i.e., the last state of our trajectory).

Note that such hidden states encodes some information about our sequence which will help in predicting the next uncertainty tubes. As the network is intended to be used in a sampling-based algorithm, where local trajectories are

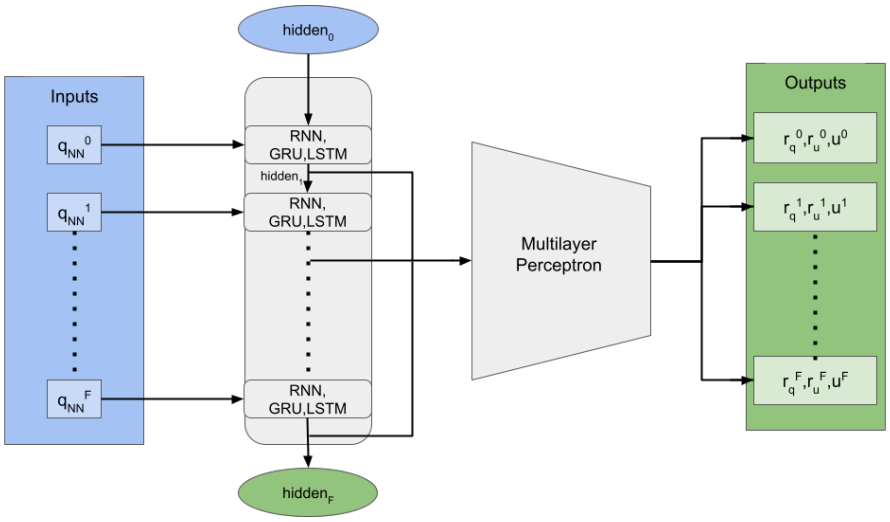

Figure 2: Representation of the neural network architecture applicable to RNN, GRU or LSTM. Blue blocks correspond to the inputs composed of a sequence of states $\boldsymbol{q_{NN}^k}$ and an initial hidden state ($hidden_0$). Green blocks refer to the output, which is a sequence of radii and control inputs ($\boldsymbol{r_q}^k, \boldsymbol{r_u}^k, \boldsymbol{u}^k$), and the final hidden state ($hidden_F$).

concatenated to form a global one, the final hidden state ($hidden_F$) is intended to be reused as the initial hidden state ($hidden_0$) in future predictions. Therefore, in general, this initial hidden state is not the null matrix.

Finally, the core of our network is composed of a single-layer GRU/ LSTM/ RNN block that feed our 'decoder' block, which is a multi layer perceptron (MLP) composed of several linear layers (see Sec.4).

The prediction process flows as follow: At $k = 0$, the first state in the input sequence $\boldsymbol{q_{NN}^0}$ and the initial hidden state $hidden_0$ are given to a single-layer GRU/LSTM/RNN block, which outputs an updated hidden state $hidden_1$. The latter is then fed to a MLP, each layer followed by a ReLU activation function except the final one, to obtain the predicted control inputs $\boldsymbol{u}^0$, the state uncertainty tubes $\boldsymbol{r_q}^0$ as well as the control uncertainty tubes $\boldsymbol{r_u}^0$. The updated hidden state h_1 is then given back to the GRU/LSTM/RNN block along with the second element of the input sequence. This process is repeated for each element $\boldsymbol{q_{NN}^k}$ until all predictions are obtained.

In the case of our quadrotor, we choose the desired states as input parameters to the GRU, as they form the input to the control loop (see Eq. 7). However, in order to make the learned model independent of workspace boundaries used during planning (i.e. robot position and initial orientation), only the desired linear velocities $\boldsymbol{v_d}$, accelerations $\boldsymbol{a_d}$, and yaw angular velocities ($\dot{\Psi}_d$), are kept as the network input components. Hence, the input to the neural network is a vector $\boldsymbol{q_{NN}^k} = [\boldsymbol{v_d}, \boldsymbol{a_d}, \dot{\Psi}_d]^T$, where k refers to the k-th state of the desired

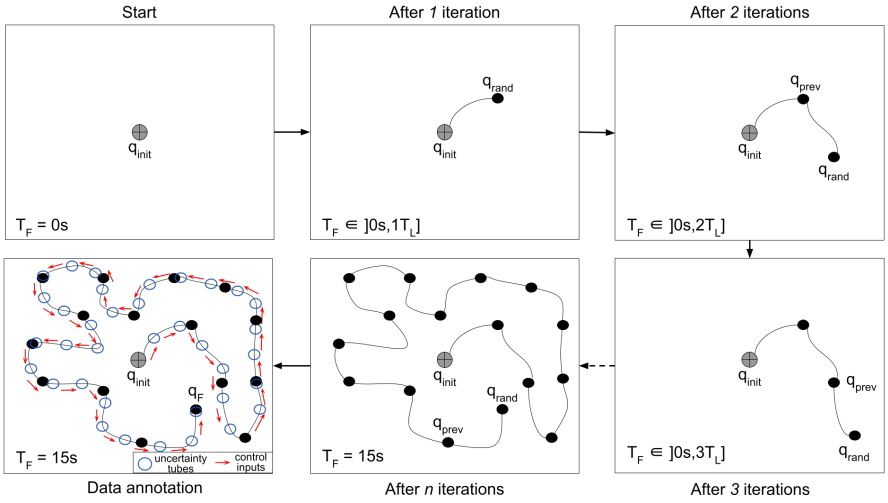

Figure 3: Dataset generation process. At each iteration, a new state $\boldsymbol{q_{rand}}$ is randomly sampled from a uniform distribution and connected to the previous connected state $\boldsymbol{q_{prev}}$ using a steering method. If the local trajectory (between $\boldsymbol{q_{rand}}$ and $\boldsymbol{q_{prev}}$) execution time is greater than 1s, it is truncated to 1s. Similarly, when the total trajectory time (T_F) exceeds 15s after adding a new state, we truncate at 15s. After n iterations, a trajectory with a total execution time of 15s made up of 1s sub-trajectories is generated. Finally, to annotate the data, the uncertainty tubes and control inputs are computed by simulating the tracking of the generated trajectory under nominal parameters (i.e $\boldsymbol{p} = \boldsymbol{p}_c$).

trajectory. As for the output vector, it is composed of the predicted control input values $\boldsymbol{u} = [u_1, u_2, u_3, u_4]^T$, the uncertainty tubes radii along the $\{x, y, z\}$-axis of the state $\boldsymbol{r_q} = [r_x, r_y, r_z]^T$, and the uncertainty tubes radii associated with the control inputs of the system $\boldsymbol{r_u} = [r_{u1}, r_{u2}, r_{u3}, r_{u4}]^T$.

4 Results

4.1 Dataset

In order to train the proposed neural network, we generated a dataset of trajectories computed in an obstacle-free environment as depicted in Figure.3. This ensures that the resulting learned model is totally independent of the environment and only depends on the system. Each global trajectory starts from the same initial hovering state ($\boldsymbol{q}_{init}$) initialized at zero velocities and accelerations[2]. Based on the same principle as a sampling-based planner, the global trajectory is made up of local sub-trajectories until a total execution time length (T_F)

[2]Note that this initialization does not hurt the generalizability of our model to different initial conditions and environments with obstacles.

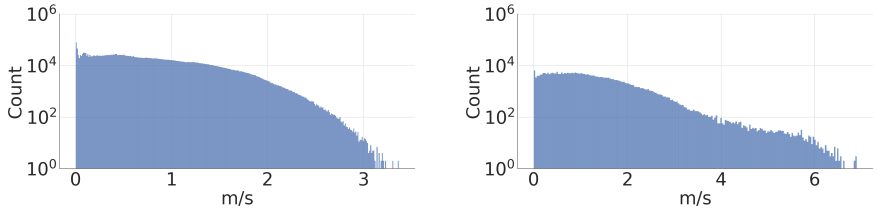

Figure 4: Velocity norm distribution in the training (left) and test sets (right).

of 15s is reached. Each local sub-trajectory is generated by uniformly sampling an arrival state ($\boldsymbol{q}_{rand}$) and connecting it to the previous sampled state ($\boldsymbol{q}_{prev}$) using a *Kinosplines* steering method [5]. These kinosplines have the advantage of enforcing the following kinodynamic constraints on the generated splines $[v_{max}, a_{max}, j_{max}, s_{max}]$, where $v_{max}, a_{max}, j_{max}$ and s_{max} represent the maximum allowed velocity, acceleration, jerk and snap respectively. If the local trajectory (between $\boldsymbol{q_{rand}}$ and $\boldsymbol{q_{prev}}$) expected execution time is greater than a maximum local duration T_l, it is truncated to T_l. Similarly, when the total trajectory duration T_F exceeds 15s, it is truncate at 15s.

To generate the outputs, i.e. to annotate the data with uncertainty tubes and control inputs, the closed-loop dynamic and the sensitivity matrices are computed by simulating the tracking of the global trajectories using an integration time step ΔT which corresponds to the same time step used for collision checking in a sampling-based motion planner.

Using this mechanism, a training and validation sets respectively composed of 8.000 and 2.000 trajectories were generated considering a maximum local duration of $T_l = 1s$ and an integration time step $\Delta T = 0.05s$, making sure that every trajectory in the dataset is different.

In order to show the reliability and generalizability of the learned model, a test set composed of 1.000 trajectories was generated in the same way, but considering a maximum local duration $T_l = 2s$. As a result, trajectories with higher velocities are encountered in the test set compared to the validation set, as depicted in Figure 4 where velocity norms can reach up to 7 m.s^{-1} in the test set, compared with only 3 m.s^{-1} in the validation set[3].

The kinodynamic constraints enforced on the generated splines are $[v_{max}$, $a_{max}, j_{max}, s_{max}] = [5.0\, m.s^{-1}, 1.5\, m.s^{-2}, 15.0\, m.s^{-3}, 30.0\, m.s^{-4}]$. The data annotation was performed by computing and integrating (Eq. 6) and (Eq. 3) by mean of the dopri5 [6] ODEs solver along a desired trajectory. Once $\boldsymbol{\Pi}$ and $\boldsymbol{\Theta}$ had been computed, a simple projection was performed to recover the tubes thanks to (Eq. 5). Note that the control inputs $\boldsymbol{u}$ are computed during the ODEs resolution. The mean and standard deviation values of the various com-

[3]Note that the velocity norms exceed the velocity limit v_{max}, this is expected since this limit applies to the components of the velocity vector rather than the norm.

ponents of the output vector for the validation and test sets generated by this setup are provided in Table.1.

Output	**Validation set**	**Test set**
$\boldsymbol{r_q}$	$1.0e^{-1} \pm 2.0e^{-2}$	$1.1e^{-2} \pm 2.1e^{-2}$
$\boldsymbol{u}$	12469.3 ± 861.6	12476.8 ± 1016.5
$\boldsymbol{r_u}$	7782.6 ± 3172.3	7828.6 ± 2845.7

Table 1: Mean and standard deviation of the output vector components norm after data annotation for the validation and test sets. $\boldsymbol{r_q}$ is expressed in m, and $(\boldsymbol{u}, \boldsymbol{r_u})$, are squared propeller speeds [(rad/s)$^{\mathbf{2}}$].

4.2 Training

	RNN	GRU	LSTM
LR	$1e^{-4}$	$1e^{-3}$	$1e^{-3}$
HIDDEN	512	512	512
LINEAR	3	3	3

Table 2: Implementation details for each neural network after hyper-parameters optimization where LR refers to the learning rate, HIDDEN to the hidden state size and LINEAR to the number of linear layers.

In order to chose the most suitable model for our needs we compare 3 different models: a basic RNN, a GRU and a LSTM. Each of these is dependent on the following hyper-parameters: learning rate (LR), hidden state size (HIDDEN) and number of linear layers (LINEAR). The size of the linear layer (l_{linear}) is related to the hidden state size by letting $l_{linear} = \frac{HIDDEN}{2}$.

We optimize the hyper-parameters of the 3 models using a grid search where LR $= \{1, 1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}, 1e^{-5}\}$, HIDDEN $= \{32, 64, 128, 256, 512\}$ and LINEAR $= \{1, 2, 3\}$, and details of the best implementations found are reported in Table.2. Note that the LSTM has both a hidden state and a cell state that are the same size, so the total size of the LSTM input is 2×512.

Each model is then trained using the MSE (Mean Squared Error) loss function over 200 epochs and using 80% of the dataset generated in Sect.4.1 for the training set and the 20% left for the validation set. The outputs vector are standardize according to the mean and standard deviation of the training set, and the inputs features are min max scaled according to the maximum velocity and acceleration values of Sect.4.1.

4.3 Evaluation

The metrics chosen to quantify models performances are as follows:

Method	**Validation set**			**Test set**		
	MAE_{r_q}	MAE_u	MAE_{r_u}	MAE_{r_q}	MAE_u	MAE_{r_u}
RNN	$4.5e^{-4}$	35.8	166.4	$1.4e^{-3}$	103.3	544.8
LSTM	3.1^{-4}	17.7	**58.7**	$1.4e^{-3}$	79.7	308.1
GRU	$\mathbf{2.6e^{-4}}$	**17.3**	64.5	$\mathbf{1.1e^{-3}}$	**71.0**	**279.8**

Table 3: MAE (Mean Absolute Error) on the $\boldsymbol{r_q}$, $\boldsymbol{u}$ and $\boldsymbol{r_u}$ components of the output vector expressed, and computed on the validation and test sets for a trained RNN, GRU and LSTM model. MAE_{r_q} $[m]$ and (MAE_u, MAE_{r_u}) $[(\text{rad/s})^2]$.

Time (ms)	100 points	200 points	300 points
T_{RNN}	$\mathbf{0.8} \pm 0.5$	$\mathbf{1.7} \pm 2.7$	$\mathbf{2.5} \pm 1.6$
T_{GRU}	2.3 ± 0.4	4.4 ± 0.3	7.3 ± 1.1
T_{LSTM}	2.8 ± 0.3	5.9 ± 0.5	8.1 ± 0.6
T_{euler}	63.9 ± 8.3	132.4 ± 9.9	261.1 ± 18.0
T_{dopri5}	251.7 ± 17.3	537.2 ± 29.8	755.6 ± 34.5

Table 4: Average prediction time (ms) over 100 predictions on trajectories composed of 100 states, 200 states and 300 states, for an RNN, GRU, LSTM, an ODE Euler integrator, and the ODE dopri5 integrator.

- The MAE (Mean Absolute Error) of the norm of the different output components $\boldsymbol{r_q}$, $\boldsymbol{u}$ and $\boldsymbol{r_u}$, denoted $\mathbf{MAE_{r_q}}$, $\mathbf{MAE_u}$ and $\mathbf{MAE_{r_u}}$ respectively. We then compute $\mathbf{MAE_{\boldsymbol{u}}}$ as:

$$\mathbf{MAE_{\boldsymbol{u}}} = MAE(\|\boldsymbol{u}\|, \|\hat{\mathbf{u}}\|) \tag{8}$$

  The same goes for $\mathbf{MAE_{r_q}}$ and $\mathbf{MAE_{r_u}}$.

- Prediction time compared with traditional methods involving ordinary equation solvers (e.g. Runge Kutta 4). Inference time is denoted $T_{solver/NN}$ according to the model or solver used. The ODEs were implemented using the JiTCODE [7] module which converts the equations to be integrated into C-compiled code. The **Euler** method or the **dopri5** integrator were then used to solve each ODE and take advantage of this compiled function. All following results were obtained on an Intel i9 CPU@2.6GHz processor with one RTX A3000 GPU.

Table 3 reports the MAE for the different output vector components on the validation and test sets. First of all, we observe that **RNN** offers the least accurate predictions, with up to 7% error on the $\boldsymbol{r_u}$ components of the test set. On the other hand, **GRU** provides the best accuracy on all the components on both sets except for $\boldsymbol{r_u}$ on the validation set, but for which predictions remain close to expected values with less than 1% deviation from expected mean values. **GRU** shows the best reliability and generalizability of the trained model to

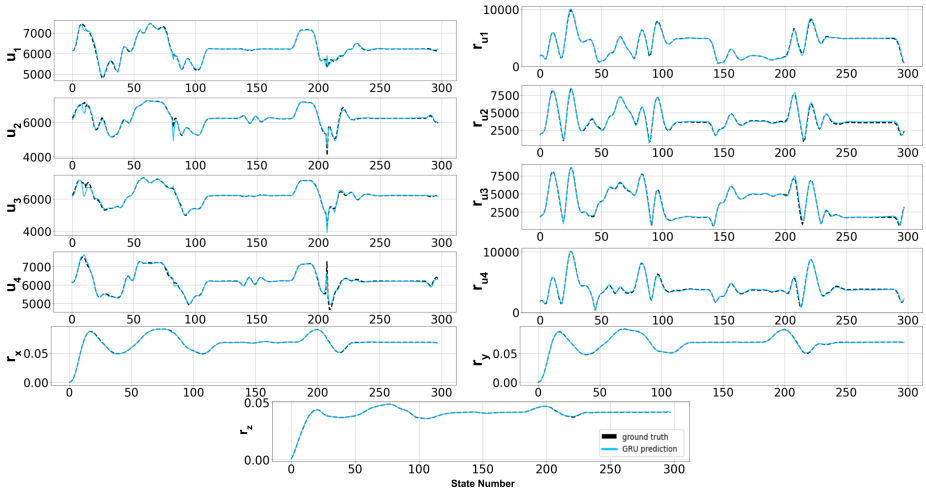

Figure 5: Example of **GRU** predictions on a 300-state trajectory of the test set. Predicted outputs are displayed in blue against true values in black. r_x, r_y, r_z are expressed in m, and control input associated values (u_i, r_{ui}) are squared propeller speeds [(rad/s)^2].

unseen samples from a different distribution. **GRU** performance on the test set shows that the predictions along $\boldsymbol{r_q}$ remain highly accurate (less than 1 millimeter average error) and that the highest errors are obtained on $\boldsymbol{r_u}$ but do not exceed 4% of the expected average value. The **LSTM** provides the best predictions on the $\boldsymbol{r_u}$ component of the validation set. However, the results show a slightly lower accuracy than **GRU** on the other components with an average error of 4.5% on test set $\boldsymbol{r_u}$ predictions. Nevertheless, **LSTM** is more accurate on all components than **RNN**. Overall, **RNN** performs the worst while **GRU** and **LSTM** provide similar results, with an overall better accuracy for **GRU**. Moreover, the latter shows a better generalization to unseen trajectories that are slightly different from the training set, as illustrated in Fig. 5 where higher velocities than in the training set are encounter. In addition, Fig. 6 shows predicted vectors for a 600-state trajectory generated in the same way as the test set but considering a total length of 30s (i.e. $T_F = 30s$), thus demonstrating the stability of the proposed method.

In order to choose the most advantageous model or method for computing uncertainty tubes in terms of prediction time, the methods/models are applied to trajectories of different lengths (i.e. made up of a certain number of desired states). Table 4 shows the results obtained on trajectories of several hundred states. Note that with the current system, the number of ordinary equations solved for each element in the sequence (i.e. trajectory state) is equal to 91 (see Sect. 2). We observe that the greater the length of the trajectory to be integrated, the greater the gap between ODEs solver methods and neural networks in average prediction time. Results show that as the number of states in the tra-

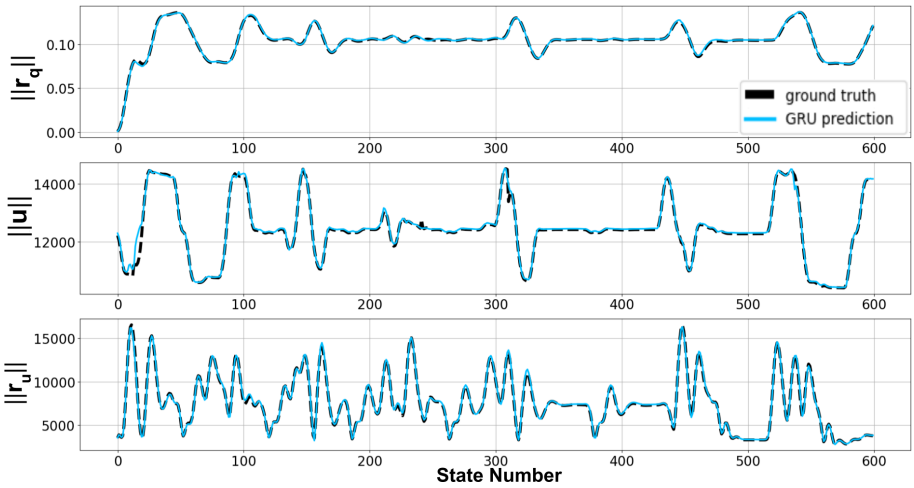

Figure 6: Example of **GRU** predictions on a 600-state trajectory generated in the same way as the test set. $||\boldsymbol{r_q}||, ||\boldsymbol{u}||$ and $||\boldsymbol{r_u}||$ refer to the norm of their respective vector. Predicted outputs are displayed in blue against true values in black. $||\boldsymbol{r_q}||$ is expressed in m, and control input associated values ($||\boldsymbol{u}||, ||\boldsymbol{r_u}||$) are squared propeller speeds [$(\text{rad/s})^{\mathbf{2}}$].

jectory increases, the time gain reaches two orders of magnitude using recurrent neural network architectures. We also note that among these models, **RNN** is the fastest to perform the predictions, which is explained by the fact that the network size is smaller than **LSTM**, and the **RNN** cell performs fewer internal operations than the **GRU** cell. In a robust sampling-based motion planning context, the neural network is meant to be queried tens of thousands of times on multi-states trajectories. Hence, these observed gains can be exponential depending on the number of trajectories to be evaluated.

Finally, even if **RNN** excels in inference time when making predictions; its accuracy is noticeably lower when contrasted with that of **GRU**. As for the **LSTM**, it provides the slowest inference time and less accurate predictions than the **GRU**, except for the $\boldsymbol{r_u}$ component of the validation set. Additionally, its hidden state implementation results in twice as many parameters having to be saved per sampling-based motion planner iteration compared to **GRU**, which translates by a higher memory cost when growing trees or graphs with thousands of nodes. Therefore, based on the models implementation details and results presented above, the use of **GRU** is recommended in the context of a sampling-based motion planner. Indeed, the latter is much faster than methods based on ODEs solvers while offering the most accurate predictions, thus offering the best trade-off between inference time and accuracy.

5 Conclusion

We have presented a **GRU**-based neural network architecture to predict uncertainty tubes and control inputs along sequences of desired states for any dynamical system. Results on a quadrotor use case show that leveraging recurrent neural network architectures is of key importance due to the temporal dependency of the predictions. Furthermore, we have shown that a **GRU** is more appropriate in a sampling-based tree planner context than **RNN** or **LSTM** as it proposes the best compromise between prediction accuracy, generalizability, inference time and memory cost.

References

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

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

[3] P. Brault, "Robust trajectory planning algorithms for robots with parametric uncertainties," Ph.D. dissertation, Université de Rennes, 2023.

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

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

[6] J. R. Dormand and P. J. Prince, "A family of embedded runge-kutta formulae," vol. 6, no. 1. Elsevier, 1980, pp. 19–26.

[7] G. Ansmann, "Efficiently and easily integrating differential equations with JiTCODE, JiTCDDE, and JiTCSDE," vol. 28, no. 4, 2018, p. 043116.