ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA241016

Learning Uncertainty Tubes via Recurrent Neural Networks for Planning Robust Robot Motions

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

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

Abstract. Taking into account the effects of parameter uncertainties in the robot model is crucial to the robustness of motion generation. One approach to address this issue is to compute 'uncertainty tubes' enveloping the robot state for any combination of parameters within a given range, and to use these tubes to robustly check for collisions within a motion planning algorithm. However, computing these tubes for complex dynamical systems can be too computationally expensive due to the need to solve and integrate potentially numerous non-linear ordinary differential equations (ODEs) associated with robot dynamics. To overcome this limitation, we propose a GRU-based architecture that provides fast and accurate estimation of these uncertainty tubes. We demonstrate that GRUs achieve the best compromise between prediction accuracy, prediction time, and network size compared to basic RNNs and LSTMs, justifying our choice. Finally, we showcase the efficiency of the learning process within a motion planning framework for an aerial vehicle.

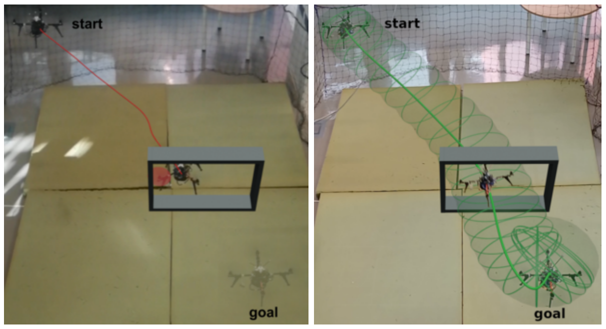

Figure 1: Execution by a real perturbed quadrotor of planned trajectories without (left) and with (right) uncertainty tubes prediction. A virtual collision is observed in the case where tubes are not used, which is not the case when predicted uncertainty tubes are used.

1 Introduction

Robust motion planning is an important and difficult problem to solve in order to guarantee the safe execution of robot motions. Indeed, in the presence of parametric uncertainties, deviations between desired and actual robot trajectories are inevitable, resulting in imperfect trajectory tracking. Neglecting these deviations at the motion planning level may result in collisions with the environment or saturation of the robot control inputs during execution.

In order to plan robust motions that take these deviations into account, some algorithms, such as the RandUP-RRT [26, 16], simulate several times the robot dynamics in the presence of random uncertainty in the robot parameters model to approximate the set of states that can be reached by the system. However, these probabilistic approaches cannot guarantee formal robustness for all uncertain parameter values within a given range. To overcome this limitation, several recent approaches [25, 18, 7, 28, 23] rely on so-called 'uncertainty tubes' that, given a range on the parametric uncertainties, bound the system state evolution over time. These uncertainty tubes, which are derived from various metrics, are used during motion planning to robustly check collisions and control inputs saturation. Nevertheless, these methods suffer from high computational cost.

In this work, we focus on uncertainty tubes computations based on the *closed-loop sensitivity* [21, 4], as it has the advantage of being applicable to any robot/controller compared to other methods. These uncertainty tubes were combined to an RRT [14] sampling-based tree planner to perform robust collisions and saturations checking within a motion planning framework named SAMP [25]. However, this framework needs to perform a new tube computation at each new iteration which requires repeatedly solving tens to hundreds of ordinary differential equations (ODEs). As thousands of iterations are required, this process quickly becomes tedious and highly time consuming, resulting in prohibitively high planning times.

Previous works have proposed learning methods to estimate safe control inputs [7, 27, 19, 29] or predict uncertainty tubes [9]. However, they are limited to a specific dynamical system (e.g., [27]) or a specific control strategy (e.g., MPC [29, 9]). Moreover, they are presented as applicable in an online manner at the control level only, making them hard to use in a global motion planning context.

Recurrent neural network architectures are well known for their excellent application to temporal data sequences, and more specifically in this case, to trajectories. Their application in motion planning frameworks has rapidly grown in recent years, e.g., to predict trajectories in sampling-based algorithms [17, 20]. Furthermore, because of their time-series nature and their ability to handle dependencies between time steps, they can take advantage of the ODEs structure [5] or approximate their solutions directly [10].

In this paper, we aim at leveraging recurrent neural networks to estimate uncertainty tubes and control inputs,hence avoiding the need for solving ODEs. We propose a multi-task neural network based on the Gated Recurrent Unit (GRU) [6] architecture which approximates these uncertainty tubes given a sequence of past desired states of any dynamical system, while focusing on a quadrotor robot as a use case. We demonstrate the performance of the proposed model in terms of

[1] Equal contribution.

accuracy and inference time, and compare it to other baselines and traditional methods (e.g. [8]) for solving the ODEs. Finally, we provide a qualitative demonstration of a real scenario illustrated by Figure 1 where a quadrotor is able to safely navigate through a narrow window thanks to the predicted tubes. To the best of our knowledge, our approach is the first learning-based method for simultaneously predicting parametric uncertainty tubes and control inputs, applicable to very diverse types of robots and controllers.

2 Closed-Loop Sensitivity

Consider an arbitrary dynamical system with a set of uncertain parameters $\boldsymbol{p} \in \mathbb{R}^{n_p}$ (i.e model parameters that are difficult to evaluate or likely to vary during execution). Let $\boldsymbol{q} \in \mathbb{R}^{n_q}$ be the system's state vector. Given any tracking controller $\boldsymbol{\eta}$ and a desired state $\boldsymbol{q}_d$ to track, the system's dynamics can be defined as follows:

$$\begin{cases} \dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}, \boldsymbol{p}), \quad \boldsymbol{q}(t_0) = \boldsymbol{q}_0, \\ \boldsymbol{u} = \boldsymbol{\eta}(\boldsymbol{q}, \boldsymbol{q_d}, \boldsymbol{p}_c, \boldsymbol{k}_c, t) \end{cases} \tag{1}$$

where $\boldsymbol{u} \in \mathbb{R}^{n_u}$ is the vector of control inputs, $\boldsymbol{p}_c$ is the vector of "nominal" parameters, i.e. the estimated nominal values of $\boldsymbol{p}$, and $\boldsymbol{k}_c \in \mathbb{R}^{n_k}$ are suitable controller gains.

Following [21, 4], let $\boldsymbol{\Pi}$ be the *state sensitivity matrix*, and $\boldsymbol{\Theta}$ the *input sensitivity matrix* such that:

$$\boldsymbol{\Pi}(t) = \left.\frac{\partial \boldsymbol{q}(t)}{\partial \boldsymbol{p}}\right|_{\boldsymbol{p}=\boldsymbol{p}_c} \qquad \boldsymbol{\Theta}(t) = \left.\frac{\partial \boldsymbol{u}(t)}{\partial \boldsymbol{p}}\right|_{\boldsymbol{p}=\boldsymbol{p}_c} \tag{2}$$

These quantities allow the quantification of how variations of the uncertain model parameters $\boldsymbol{p}$ affect the evolution of the system in closed-loop. Depending on the chosen controller, these matrices generally do not have a closed-form expression but can be computed according to the following dynamics (see [21, 4] for more details):

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

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. $\Delta \boldsymbol{q} \approx \boldsymbol{\Pi}(t)\Delta \boldsymbol{p}$, the *uncertainty tube* around $\boldsymbol{q}$ is defined by a radius $r_{q_i}(t)$ along each i-th component of the state, which bounds the state evolution $q_i(t)$ around the desired state [2] $q_{d_i}(t)$ over time:

$$q_{d_i}(t) - r_{q_i}(t) \leq q_i(t) \leq q_{d_i}(t) + r_{q_i}(t). \tag{4}$$

Let the kernel of $\boldsymbol{\Pi}$ be:

$$\boldsymbol{K}_{\boldsymbol{\Pi}}(t) = \boldsymbol{\Pi}(t)\boldsymbol{W}\boldsymbol{\Pi}(t)^T \tag{5}$$

where $\boldsymbol{W}$ is a diagonal matrix where the diagonal elements are the components of $\delta \boldsymbol{p}$. The radius $r_{q_i}(t)$ can be obtained by projecting $\boldsymbol{K}_{\boldsymbol{\Pi}}(t)$ along the i-th component of the state $\boldsymbol{q}$ (see [3] for details). Note that similar tubes can be obtained for the control inputs $\boldsymbol{u}$ using the input sensitivity matrix $\boldsymbol{\Theta}$.

Such radii computation relies on the knowledge of $\boldsymbol{\Pi}(t)$ and $\boldsymbol{\Theta}(t)$ which are computed numerically by integrating the dynamics of (Eq. 1) and (Eq. 3), and then performing the projection of (Eq. 5) in both the state and input spaces. Depending on the number of parameters and the dimensions of the state and input spaces, this computation may be too time consuming when planning robust motions of complex dynamical systems.

[2] In practice, such tubes are centered around the nominal states $\boldsymbol{q}_n$ (i.e. the state of the tracked trajectory without considering parameters perturbations, meaning $\boldsymbol{p} = \boldsymbol{p}_c$), but for simplicity we consider $\boldsymbol{q}_d \approx \boldsymbol{q}_n$.

In this paper, we present a method for improving the efficiency of this process for any dynamical system which we illustrate on a quadrotor case. In the latter, the state vector is $\boldsymbol{q} = [\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\rho}, \boldsymbol{\omega}] \in \mathbb{R}^{13}$ 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 quadrotor position and velocity vectors. The body orientation is represented by the unitary quaternion denoted with $\boldsymbol{\rho}$, while its angular velocity is $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z] \in \mathbb{R}^3$. The vector of uncertain model parameters is $\boldsymbol{p} = [m, x_{cx}, x_{cy}, J_x, J_y, J_z]^T \in \mathbb{R}^6$, with m the quadrotor mass, $x_{cx,y}$ a shift of the system center of mass along the x,y-axis, and $J_{x,y,z}$ the main inertia coefficients.

The controller considered is the Lee (or geometric) controller [15] where the control inputs $\boldsymbol{u} = [u_1, u_2, u_3, u_4]^T \in \mathbb{R}^4$ are the squared propeller angular velocities. This controller computes the control inputs according to a given desired state denoted $\boldsymbol{q_d} = [\boldsymbol{x_d}, \boldsymbol{v_d}, \boldsymbol{a_d}, \Psi_d, \dot{\Psi}_d] \in \mathbb{R}^{11}$ respectively composed of the desired positions, velocities, accelerations, yaw orientation angle, and yaw angular velocity[3]. Note that in our quadrotor application, $\boldsymbol{\Pi} \in \mathbb{R}^{13\times 6}$ and $\boldsymbol{\Theta} \in \mathbb{R}^{4\times 6}$. As a result, the computations necessary to find the uncertainty tubes involve solving nearly hundred ordinary differential equations per state.

3 Method

3.1 Problem statement

The goal is to train a neural network to approximate sensitivity-based uncertainty tubes, hence avoiding the computational cost of solving many ODEs. Moreover, we aim at predicting the control inputs that the system will exert to successfully track a desired trajectory, in addition to the uncertainty tubes on these control inputs. Given a sequence of desired robot state vectors $\mathbf{M} = \{\boldsymbol{q}_d^0, \boldsymbol{q}_d^1, ..., \boldsymbol{q}_d^n\}$ representing the desired robot's motion (i.e. a desired trajectory to follow), the task at hand is to learn a function $\boldsymbol{g}$ that estimates the radii of uncertainty tubes for each state $\boldsymbol{q}_d^k$ as well as the robot control inputs at this state such that:

$$\{\boldsymbol{r_q}^k, \boldsymbol{r_u}^k, \boldsymbol{u}^k\} = \boldsymbol{g}(\boldsymbol{q}_d^0, \boldsymbol{q}_d^1, ...\boldsymbol{q}_d^{k-1}) \tag{6}$$

Since evaluating the function $\boldsymbol{g}$ on $\boldsymbol{q}_d^k$ depends on all the previous states of the robot in $\mathbf{M}$, recurrent neural network architectures are a good fit given their ability to encode and accumulate temporal information while keeping inference time low.

3.2 Neural network architecture

We propose a multi-task learning neural network based on a GRU architecture, which takes as input the desired states of the dynamical system and outputs the control inputs as well as the uncertainty tubes around the states and the control values. A representation of our neural network architecture is presented in Figure 2. Blue blocks refer to the inputs of the model which are composed of an initial hidden state h_0 and of a sequence of desired robot states. For the quadrotor case, in order to make the learned model independent of workspace boundaries used during planning (i.e. robot position and orientation bounds) and initial robot orientations, only the desired linear velocities, accelerations, and yaw angular velocities ($\dot{\Psi}_d$), are kept as the network input components. Hence, the input to the neural network is a vector $\boldsymbol{q_{in}^k} = [\boldsymbol{v_d}, \boldsymbol{a_d}, \dot{\Psi}_d]^T$, where k refers to the k-th state of the desired trajectory.

[3] The quadrotor system being under-actuated (i.e not all components are controllable), the desired state $\boldsymbol{q}_d$ differs from the system state $\boldsymbol{q}$ (see [15] for more details).

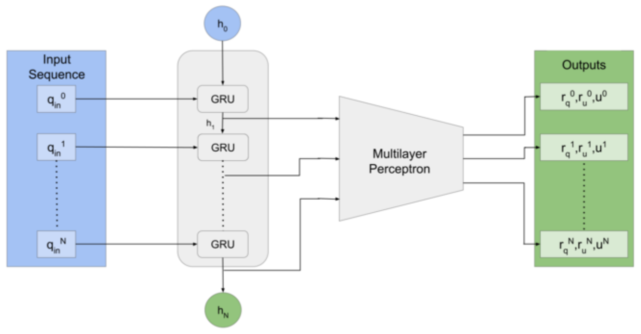

Figure 2: Representation of the proposed neural network architecture.

The outputs of the neural network correspond to the green blocks. In the case of a quadrotor, it is composed of the predicted 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 $\boldsymbol{r_u} = [r_{u1},\ r_{u2},\ r_{u3},\ r_{u4}]^T$. Moreover, the control input values $\boldsymbol{u} = [u_1,\ u_2,\ u_3,\ u_4]^T$ are required for the motion planning algorithm, and their computation depends on ODE forward integration (as shown in (Eq. 1)). Since our approach aims at eliminating the need for solving ODEs, we need to train the neural network to also predict the control inputs. Finally, h_N corresponds to the hidden state at the last point of our sequence (i.e., the last trajectory state).

At $k = 0$, the first state in the input sequence $\boldsymbol{q_{in}^0}$ and the initial hidden state h_0 are given to a single-layer GRU block with a hidden state size of 512, which outputs an updated hidden state h_1. The latter is then fed to a 3-layer multi-layer perceptron (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 block along with the second element of the input sequence. This process is repeated for each element $\boldsymbol{q_{in}^k}$ until all predictions are obtained.

The network is intended to be used in a sampling-based tree planner, where local trajectories are concatenated to form a global one. Thus, since the hidden state encodes and accumulates temporal information about the input sequence, the final hidden state h_N of a local trajectory obtained after an iteration of a sampling-based tree planner can be saved and then given back as the initial hidden state h_0 for future local trajectories considered during the next iterations. Therefore, in practice, this initial hidden state is generally not null.

3.3 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[4]. 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) 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 [2]. These kinosplines have the advantage of enforcing the following kinodynamic constraints on the generated splines

[4] Note that this initialization does not hurt the generalizability of our model to different initial conditions and environments with obstacles (cf. Sect. 6).

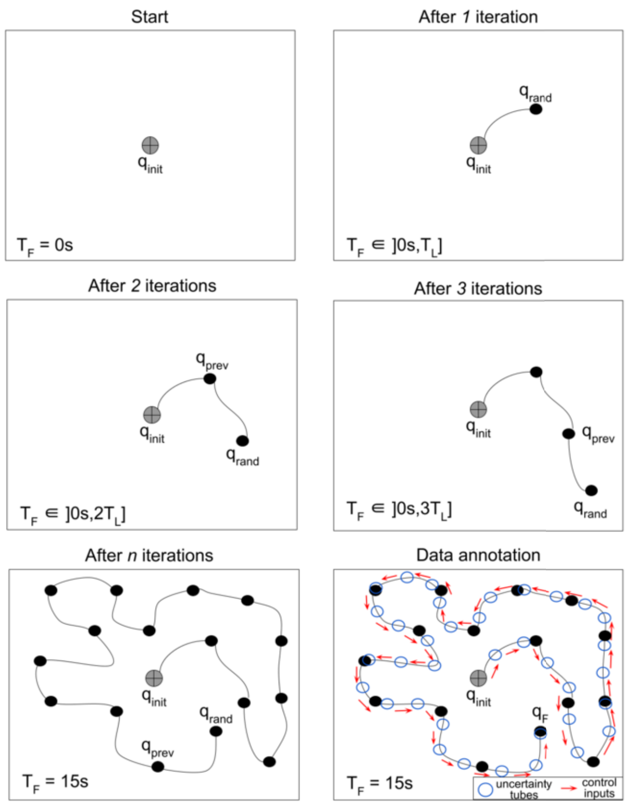

Figure 3: Dataset generation process. Starting from the hovering state $\boldsymbol{q}_{init}$, at each iteration a new state $\boldsymbol{q_{rand}}$ is randomly sampled and connected to the previous state $\boldsymbol{q_{prev}}$ until a total trajectory length (T_F) of 15s is reached. Data annotation is performed by simulating the tracking of the generated trajectory under nominal parameters.

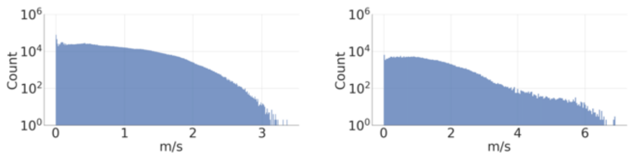

Figure 4: Velocity norm distribution in the training (left) and test sets (right).

$[v_{max}, a_{max}, j_{max}, s_{max}]$, 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 after adding a new state, 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 set composed of 8.000 trajectories and a validation set composed of 2.000 trajectories were generated, making sure that every trajectory in the dataset is different. These trajectories were generated considering a maximum local duration of $T_l = 1s$ and an integration time step $\Delta T = 0.05s$.

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 nominal values of the uncertain parameters presented in Sect. 2 are $\boldsymbol{p}_c = [1.113, 0.0, 0.0, 0.015, 0.015, 0.007]^T$ and their associated uncertainty range used for the tubes computation are $\delta\boldsymbol{p} = [7\%, 3cm, 3cm, 10\%, 10\%, 10\%]^T$, which represents the variation of the parameters w.r.t. their associated nominal value. The controller gains used are $\boldsymbol{k_x} = [20.0\,, 20.0\,, 25.0]$, $\boldsymbol{k_v} = [9.0\,, 9.0\,, 12.0]$, $\boldsymbol{k_R} = [4.6\,, 4.6\,, 0.8]$, $\boldsymbol{k_\omega} = [0.5\,, 0.5\,, 0.08]$.

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[5].

The data annotation was performed by computing and integrating (Eq. 1) and (Eq. 3) by mean of the dopri5 [8] 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 components 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 angular velocities [(rad/s)²].

4 Experiments

4.1 Training

Before training the neural network, the input features of the dataset (train, val, and test) were min-max scaled according to the maximum velocity and acceleration values used to generate the kinosplines of Sect.3.3 as it is guaranteed that the velocities and accelerations generated cannot exceed these values. This ensures that all velocity values are in the interval $[-v_{max}, v_{max}]$, while all the acceleration values are in $[-a_{max}, a_{max}]$. The annotations were also normalized using a standard scaling based on the mean and standard deviation values of the training set annotations.

After data normalization, the neural network was trained for 200 epochs using the Adam optimizer [13] with a learning rate of $1e^{-3}$ and a batch size of 128. It was trained using the MSE (Mean Squared Error) loss function and evaluated continuously on the validation set. The weights achieving the best performance on the validation set were saved and used during our experiments.

4.2 Evaluation

In order to demonstrate the necessity of using a recurrent neural network due to the temporal dependencies of the predictions, we implemented a simple **MLP** baseline by substituting the recurrent layer with a single-layer linear encoder which takes as input a single element in the sequence and outputs its corresponding predictions.

Also, in order to justify the choice of a **GRU** architecture, we compare our proposed model to two other versions which replace the GRU block by a basic RNN [22] and an LSTM [11] block respectively. Note that we do not report the comparison with Transformers [24] as they do not benefit from the hidden state when starting predictions from non zero values, and poorly generalize to longer sequences than those in the training set. The **RNN** and **LSTM** models were trained with a hidden state size of 512, a **MLP** composed of 3 linear layers, and a learning rate of $1e^{-4}$ for the **RNN** case against $1e^{-3}$ for the **LSTM** model. Note that an LSTM layer is composed of a hidden state and a cell state (contrarily to GRUs and RNNs which have a hidden state only), so the real LSTM latent state size is 2×512. All used hyper-parameters were validated using a grid search.

Finally, in order to measure the computational cost gain achieved by our method and compare the inference time of the neural network to traditional methods, we implemented two known ODEs integrators which we applied to uncertainty tubes computation. The first is **dopri5**, which leverages the Runge Kutta-4 algorithm to numerically approximate the solutions to ODEs. The second is the **Euler** method, which is known to be faster but yields less accurate results.

4.3 Metrics

We evaluate the performance of our model using the Mean Absolute Error (MAE) over the different outputs. Rather than comparing the results on each sub-component of the input (e.g $\{x, y, z\}$ for r_q), we combine them into a single metric in order to obtain simpler and more general comparisons, as follows:

- $\mathbf{MAE}_{\boldsymbol{r_q}}$: represents the mean absolute error on the norm of $\boldsymbol{r_q}$. For a given datapoint, given the ground truth and predicted state uncertainty tubes $\boldsymbol{r_q} = [r_x, r_y, r_z]^T$ and $\hat{\mathbf{r}}_{\mathbf{q}} = [\hat{r}_x, \hat{r}_y, \hat{r}_z]^T$, we compute the norms $\|\boldsymbol{r_q}\|$ and $\|\hat{\mathbf{r}}_{\mathbf{q}}\|$. We then compute $\mathbf{MAE}_{\boldsymbol{r_q}}$ as:
$$\mathbf{MAE}_{\boldsymbol{r_q}} = MAE(\|\boldsymbol{r_q}\|, \|\hat{\mathbf{r}}_{\mathbf{q}}\|) \quad (7)$$
This metric is expressed in meters (m).
- $\mathbf{MAE}_{\mathbf{u}}$: represents the mean absolute error on the norm of $\boldsymbol{u}$. We compute the norms of the ground truth and predicted control inputs $\|\boldsymbol{u}\|$ and $\|\hat{\boldsymbol{u}}\|$. We then compute $\mathbf{MAE}_{\mathbf{u}}$ as the mean absolute error between these two norms. It is expressed in [(rad/s)²].
- $\mathbf{MAE}_{\boldsymbol{r_u}}$: represents the mean absolute error on the norm of $\boldsymbol{r_u}$. As for the two previous metrics, $\mathbf{MAE}_{\boldsymbol{r_u}}$ is defined as the mean absolute error between the norms $\|\boldsymbol{r_u}\|$ and $\|\hat{\mathbf{r}}_{\mathbf{u}}\|$. This metric is expressed in [(rad/s)²].

These metrics are averaged over all elements of a sequence, then averaged over all sequences in the validation and test sets. In addition, we use inference time as metric in order to evaluate the computational cost of our method compared to the defined baselines, as well as traditional uncertainty tubes computation methods involving an ODE solver. Time is denoted $T_{solver/NN}$ according to the model or solver used.

4.4 Implementation details

The ODEs were implemented using the JiTCODE [1] 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. The code is available at: https://gitlab.laas.fr/swasiela/learning_sensitivity.

[5] 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.

5 Quantitative Results

5.1 Model comparison

Table 2 reports the MAE for the different output vector components on the validation and test sets. First of all, we note that the **MLP** provides a poor performance, confirming the need for a recurrent neural network. We observe up to a 30% deviation from the average expected values (cf. Table 1) on the validation and test sets.

Among recurrent models, 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 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.

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 3 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 trajectory 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.

5.2 Ablation study

In theory, all the inputs to our neural network are needed by the controller and the uncertainty tubes computation methods to obtain the desired outputs. In order to confirm this for the neural network as well, we conduct an ablation study on the **GRU** model to show that the choice of input vector components described in Sect. 3.2 is the most relevant for the quadrotor case. The components considered for the study are the desired linear and angular velocities, and the desired accelerations, denoted V and A respectively. Our proposed model takes both inputs and is denoted $\mathbf{GRU}_{\boldsymbol{VA}}$. We define two ablations, one for each input, resulting in two models: $\mathbf{GRU}_{\boldsymbol{V}}$ which takes as input the linear and angular velocities only, and $\mathbf{GRU}_{\boldsymbol{A}}$ which only takes the accelerations as input. Each model is trained in the same way detailed in Sect. 4.1 and is evaluated according to the MAE metrics described in Sect. 4.3.

Note that since the network is intended to be used in a sampling-based approach, no ablation study is performed on the output vector since our objective is to use a single model within the planner, keeping the memory and computational costs low.

Table 4 provides the results of the ablation study. Results show that each input component specializes in predicting one of the desired outputs. Specifically, the model $\mathbf{GRU}_{\boldsymbol{V}}$ using the velocity component only yields accurate results for $\boldsymbol{r_q}$ and $\boldsymbol{r_u}$ on both the validation and test sets, while the one solely using the accelerations leads to accurate results on $\boldsymbol{u}$ on both sets.

These outcomes are expected since $\boldsymbol{r_q}$ depends indirectly on the state of the robot $\boldsymbol{q}$ only, which can be inferred from the velocities easily. On the other hand, the inner workings of the chosen controller give more importance to the accelerations of the robot, while the velocities are used for computing internal errors. This explains the fact that $\mathbf{GRU}_{\boldsymbol{A}}$ yields good results for the $\boldsymbol{u}$ predictions.

On the other hand, Table 4 shows an improvement on both the control inputs $\boldsymbol{u}$ and the control uncertainty tubes $\boldsymbol{r_u}$ using both the velocities and the accelerations as inputs to the neural network. Indeed, even though a slight degradation can be noted in the performance of $\boldsymbol{r_q}$ predictions, $\mathbf{GRU}_{\boldsymbol{VA}}$ achieves the best overall results on both the validation and test sets, thus justifying our implementation of Sect. 3.2. This can be explained by the fact that the output values are not related to the input components independently, they also depend on the interactions between these variables. Also, we remind that the chosen controller and uncertainty tubes computation methods use both the velocities and the accelerations of the robot to obtain the desired outputs.

Note that the results of this ablation study are specific to the chosen case of the quadrotor/controller. Another dynamical system might need a different set of inputs to obtain accurate predictions.

6 Qualitative Results

The robust generalization of the **GRU** model to the test set is highlighted in Figure 6, depicting predictions for all the components of the output vector for a trajectory composed of 300 states (i.e. $T_F = 15s$). We observe larger prediction errors during transient phases involving the higher velocities present in the test set than in the training one. We note that the neural network predictions in these phases have a moving average 'behavior'. However, an overall good fitting of the predictions can be highlighted.

In addition to the results discussed in Sect. 5.1, Figure 7 shows an example of **MLP**'s inability to provide accurate predictions on a 300-state test set trajectory, with a tendency to simply average expected

Method	Validation set			Test set		
	MAE$_{r_q}$ (m)	**MAE**$_{u}$ $[(rad/s)^2]$	**MAE**$_{r_u}$ $[(rad/s)^2]$	**MAE**$_{r_q}$ (m)	**MAE**$_{u}$ $[(rad/s)^2]$	**MAE**$_{r_u}$ $[(rad/s)^2]$
MLP	$8.9e^{-3}$	559.6	2079.0	$9.5e^{-3}$	608.5	2050.9
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 (ours)	$\mathbf{2.6e^{-4}}$	**17.3**	64.5	$\mathbf{1.1e^{-3}}$	**71.0**	**279.8**

Table 2: MAE on the r_q, u and r_u components of the output vector computed on the validation and test sets for a trained RNN, GRU, and LSTM model.

Time (ms)	100 states	200 states	300 states
T_{euler}	63.9 ± 18.3	136.7 ± 29.9	260.5 ± 38.0
T_{dopri5}	251.7 ± 17.3	537.2 ± 29.8	755.6 ± 34.5
T_{RNN}	$\mathbf{0.9} \pm 0.3$	$\mathbf{1.8} \pm 0.3$	$\mathbf{2.2} \pm 0.5$
T_{LSTM}	2.5 ± 0.3	4.8 ± 0.3	7.7 ± 0.8
T_{GRU}(ours)	2.1 ± 0.2	4.3 ± 0.3	5.8 ± 0.4

Table 3: Average prediction time (ms) over 100 predictions on trajectories composed of 100 states, 200 states and 300 states, for an **RNN**, **GRU**, **LSTM**, the **Euler** integrator, and the **dopri5** integrator.

values over the latter. In comparison, the **GRU** model is able to accurately predict the control inputs and the uncertainty tube across the whole trajectory. We also note a close performance between **GRU** and **LSTM** on all tasks, while **RNN** lags behind and tends to underestimate large variations of control uncertainty tubes r_u.

In order to show the stability of the proposed method, Figure 5 depicts the norm of 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. $T_F = 30s$). Note that this trajectory is twice as long as those used in training, validation and test sets. The results show the same behavior in transient phases where higher velocities are encountered. However, we observe that the model is consistent throughout longer sequences even with a different distribution from the one it was trained one. This is convenient in a sampling-based motion planning context, where successive local trajectories must be evaluated. Note that, in the dataset all trajectories start from an hovering state (i.e., null velocities and accelerations). Hence, in order to generalize to trajectories where initial velocities and accelerations are not null we can simply append a planned sub-trajectory arriving at this initial state and starting from the hovering state.

Finally, to demonstrate the model's ability to predict accurate uncertainty tubes in the context of a sampling-based motion planner, we provide a motion planning scenario involving a real quadrotor,

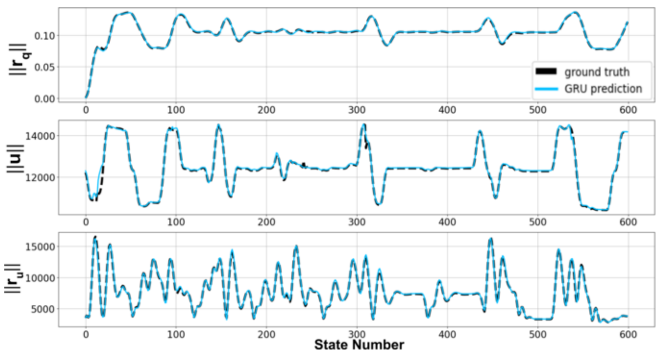

Figure 5: Example of **GRU** predictions on a 600-state trajectory generated in the same way as the test set. $||r_q||$, $||u||$ and $||r_u||$ refer to the norm of their respective vector. Predicted outputs are displayed in blue against true values in black. $||r_q||$ is expressed in m, and control input associated values ($||u||$, $||r_u||$) are squared propeller angular velocities [(rad/s)²].

with perturbed parameter values with respect to the nominal values in the model, navigating through a narrow window. In order to plan the trajectories, we used an implementation of RRT with and without tubes prediction, referred to as "non-robust" and "robust" respectively. In the latter case, the neural network was incorporated within the collision checking function, enlarging the robot shape by the predicted uncertainty tubes on the robot state r_q and checking that the predicted control inputs u remain within acceptable limits for the system using the predicted control inputs uncertainty tubes r_u.

A non-robust trajectory and a robust trajectory were planned and executed 10 times by a 'perturbed' real quadrotor. In order to disturb the system, a mass unknown to the controller was attached to the drone, so the system mass, center of mass and inertia become uncertain in the range δp of Sect. 4.2. Collisions were virtually checked over the real executions, thus mitigating the risk of real crashes and damaging the drone. Figure 1 shows an execution with virtual collision checking of a non-robust and a robust trajectory by the real quadrotor with uncertain parameters. Overall, the results show solely 85% success rate for the non-robust case against 100% for the robust one, demonstrating the ability of the proposed model to learn efficient tubes in order to plan robust motion. Furthermore, an RRT implementation using the **GRU** shows up to 5 times faster planning compared to implementations involving the traditional tubes computation methods. This gain considerably improves for optimal sampling-based tree planners such as RRT* [12], since they evaluate much more local trajectories in order to refine solutions.

7 Conclusion

In this paper 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. The application scenario involving a quadrotor demonstrates the model's ability to predict efficient uncertainty tubes in order to plan safe motions. However, although the proposed **GRU** is independent of the robot's environment, it is still dependent on the system's nominal parameters and controller gains. Future work will focus first on learning the controller's optimal gains, given that these are still hand-tuned, and then on making the model independent of the system's nominal parameters. Moreover, we plan on validating the proposed approach on other dynamic systems with different complexities.

Acknowledgements

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.

Method	Validation set			Test set		
	MAE$_{r_q}$ (m)	**MAE**$_{\boldsymbol{u}}$ $[(rad/s)^2]$	**MAE**$_{r_u}$ $[(rad/s)^2]$	**MAE**$_{r_q}$ (m)	**MAE**$_{\boldsymbol{u}}$ $[(rad/s)^2]$	**MAE**$_{r_u}$ $[(rad/s)^2]$
GRU$_V$	$\mathbf{2.5e^{-4}}$	25.2	74.6	$\mathbf{1.0e^{-3}}$	92.8	365.6
GRU$_A$	$2.6e^{-4}$	17.9	77.6	$1.3e^{-3}$	82.7	418.5
GRU$_{VA}$	$2.6e^{-4}$	**17.3**	**64.5**	$1.1e^{-3}$	**71.0**	**279.8**

Table 4: Results of the ablation study measured in term of MAE on the $\boldsymbol{r_q}$, $\boldsymbol{u}$ and $\boldsymbol{r_u}$ components of the output vector. V, A refer to the different inputs considered (linear and angular velocities, accelerations respectively). The presence of one as a subscript to the **GRU** model means that it is used as an input (e.g. $\boldsymbol{GRU_A}$ takes as input the acceleration only).

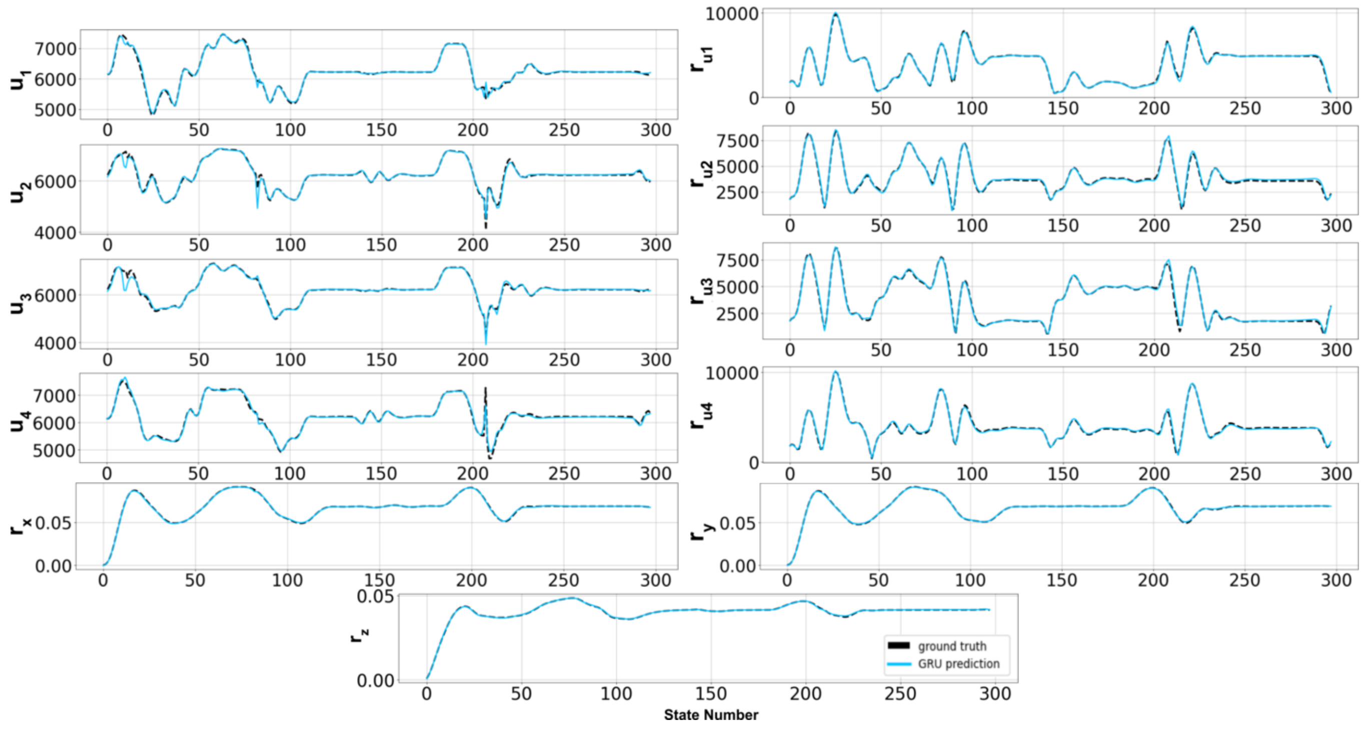

Figure 6: Example of **GRU** predictions on a 300-state trajectory of the test set. Predicted outputs are displayed in blue against true values in black. r_x, r_y, r_z are expressed in m, and control input associated values (u_i, r_{ui}) are squared propeller angular velocities [(rad/s)²].

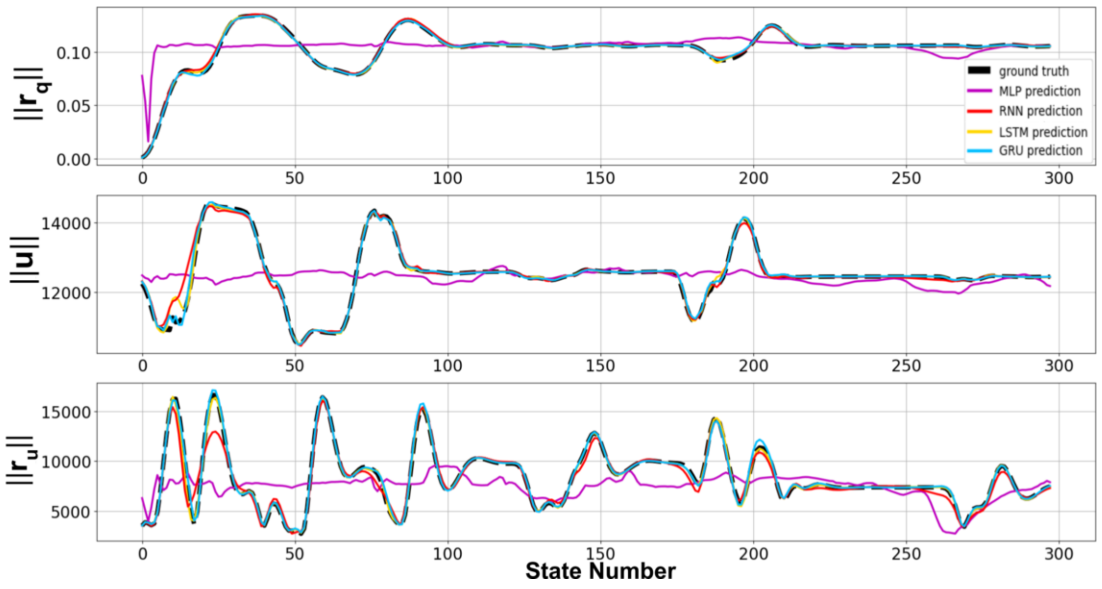

Figure 7: Comparison of predictions obtained using the different recurrent neural network architectures as well as the **MLP** on a 300-state trajectory of the test set. $||\boldsymbol{r_q}||$, $||\boldsymbol{u}||$ and $||\boldsymbol{r_u}||$ refer to the norm of their respective vector. True values are displayed in black, **GRU** predictions in blue, **MLP** predictions in purple, **LSTM** outputs in yellow, and basic **RNN** predictions are in red.

References

[1] Gerrit Ansmann, 'Efficiently and easily integrating differential equations with JiTCODE, JiTCDDE, and JiTCSDE', volume 28, (2018).

[2] Alexandre Boeuf, Juan Cortés, and Thierry Siméon, 'Motion planning'. Springer, (2019).

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

[4] Pascal Brault, Quentin Delamare, and Paolo Robuffo Giordano, 'Robust trajectory planning with parametric uncertainties', in *IEEE ICRA*, (2021).

[5] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud, 'Neural ordinary differential equations', volume 31, (2018).

[6] Kyunghyun Cho, Bart van Merriënboer, Ça$u$glar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, 'Learning phrase representations using rnn encoder–decoder for statistical machine translation', in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (2014).

[7] Glen Chou, Necmiye Ozay, and Dmitry Berenson, 'Model error propagation via learned contraction metrics for safe feedback motion planning of unknown systems', in *IEEE CDC*, (2021).

[8] John R Dormand and Peter J Prince, 'A family of embedded runge-kutta formulae', volume 6. Elsevier, (1980).

[9] David D Fan, Ali-akbar Agha-mohammadi, and Evangelos A Theodorou, 'Deep learning tubes for tube MPC', (2020).

[10] K Gajamannage, DI Jayathilake, Y Park, and EM Bollt, 'Recurrent neural networks for dynamical systems: Applications to ordinary differential equations, collective motion, and hydrological modeling', volume 33. AIP Publishing, (2023).

[11] Sepp Hochreiter and Jürgen Schmidhuber, 'Long short-term memory', volume 9. MIT press, (1997).

[12] Sertac Karaman and Emilio Frazzoli, 'Sampling-based algorithms for optimal motion planning', volume 30. Sage Publications Sage UK: London, England, (2011).

[13] Diederik P Kingma and Jimmy Ba, 'Adam: A method for stochastic optimization', (2014).

[14] Steven LaValle, 'Rapidly-exploring random trees: A new tool for path planning'. Department of Computer Science, Iowa State University, (1998).

[15] T. Lee, M. Leok, and N. H. McClamroch, 'Geometric tracking control of a quadrotor uav on se(3)', in *IEEE CDC*, (2010).

[16] Thomas Lew, Lucas Janson, Riccardo Bonalli, and Marco Pavone, 'A simple and efficient sampling-based algorithm for general reachability analysis', in *Learning for Dynamics and Control Conference*. PMLR, (2022).

[17] Yuxuan Liang, Kun Ouyang, Hanshu Yan, Yiwei Wang, Zekun Tong, and Roger Zimmermann, 'Modeling trajectories with neural ordinary differential equations.', in *IJCAI*, (2021).

[18] Anirudha Majumdar and Russ Tedrake, 'Funnel libraries for real-time robust feedback motion planning', volume 36. SAGE Publications Sage UK: London, England, (2017).

[19] Prabhat K Mishra, Mateus V Gasparino, Andres EB Velsasquez, and Girish Chowdhary, 'Deep model predictive control with stability guarantees', (2021).

[20] Ramya S Nair and P Supriya, 'Robotic path planning using recurrent neural networks', in *IEEE ICCCNT*, (2020).

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

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, 'Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986', volume 71, (1986).

[23] Sumeet Singh, Hiroyasu Tsukamoto, Brett T Lopez, Soon-Jo Chung, and Jean-Jacques Slotine, 'Safe motion planning with tubes and contraction metrics', in *IEEE CDC*, (2021).

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, et al., 'Attention is all you need', *Advances in neural information processing systems*, **30**(1), (2017).

[25] Simon Wasiela, Paolo Robuffo Giordano, Juan Cortés, and Thierry Simeon, 'A sensitivity-aware motion planner (samp) to generate intrinsically-robust trajectories', in *IEEE ICRA*, (2023).

[26] Albert Wu, Thomas Lew, Kiril Solovey, Edward Schmerling, and Marco Pavone, 'Robust-rrt: Probabilistically-complete motion planning for uncertain nonlinear systems', in *The International Symposium of Robotics Research*. Springer, (2022).

[27] Yujia Zhang, Guang Li, and Mustafa Al-Ani, 'Robust learning-based model predictive control for wave energy converters'. IEEE, (2024).

[28] Pan Zhao, Arun Lakshmanan, Kasey Ackerman, Aditya Gahlawat, Marco Pavone, and Naira Hovakimyan, 'Tube-certified trajectory tracking for nonlinear systems with robust control contraction metrics', volume 7, (2022).

[29] Tim Zieger, Hoang Hai Nguyen, Erik Schulz, Thimo Oehlschlägel, and Rolf Findeisen, 'Non-diverging neural networks supported tube model predictive control', in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, (2022).