TRAJECTORY MONITORING FOR A DRONE USING INTERVAL ANALYSIS

Sylvain LARGENT and Julien ALEXANDRE DIT SANDRETTO

***Abstract*— When modelizing a robot, uncertainties are bound to be taken into account. Uncertainties may appear because of approximations linked to the model. Sometimes uncertainties are unavoidable as they are linked to the sensors' accuracies, or inherent to the control of the robot. For instance, interval observers could be used for parameter estimation and state estimation. This paper proposes a method to consider all these uncertainties and to monitor the reliance of trajectories using interval analysis. The case study of this article is to monitor the trajectory of a holonomic drone controlled by its velocity, but the monitoring could be extended to more complex dynamic systems.**

I. INTRODUCTION

The reliance on the motion of robots and the robustness of the control is an important field of study in robotics, there have been several approaches on the problem [5], [6], [8], [14], such as stochastic approaches [15], [16]. This paper is an introductary work to the trajectory verification of a drone, using interval analysis and interval integration [1], [2]. In this case, the studied system is a drone which is governed by an ordinary differential equation of the following form:

$$\dot{y} = f(y) \text{ with } y(0) \in [y_0] \text{ and } t \in [0, t_{horizon}] \qquad (1)$$

The function $f : \mathbb{R}^n \to \mathbb{R}^n$ is the flow, $y \in \mathbb{R}^n$ is the vector of state variables, and $\dot{y}$ is the derivative of y with respect to time t [3]. The model of the drone developped in the article is a rather simplistic model, which induces a large number of uncertainties. In our case study, sensors and intervals are directly used to measure the position of the drone. But the monitoring could be coupled with an interval observer for the state estimation which could add significant uncertainties [6], [11]. The purpose of this paper is to evaluate the reliability of a trajectory while considering all those uncertainties [8], using a guaranteed integration method [4], [3]. There are several libraries developping efficient mathematical tools based on the mathematical area of interval integration. The one used in this paper is DynIbex [3].

This paper is organized as follows. Some preliminaries are given in Section II. Section III gives details about the path planner (using RRT algorithms [9], [10]) and the PI control of the drone. Section IV emphasizes the usage of interval analysis in the study of the drone. And finally Section V is devoted to the monitoring of the trajectory, and the results obtained.

The applications of this paper could be further extended with a little more time and implementation, the path planning could be extended to nonholonomic systems [10], the monitoring of the trajectory could be done in real-time with a sliding horizon of time [17], and the detection of more complex and moving obstacles could be implemented as well.

II. PRESENTATION OF THE DRONE AND ITS ENVIRONMENT

A. Concise presentation of the drone

The drone used throughout the paper is a Tello EDU drone by Ryze. Its weigh is 87g and its dimensions are 98x92.5x41 mm. The drone was programmed in ROS, the user has control over the speed of the drone (the linear speed and the angular speed). In our case study, the control was limited to translational movements and low speeds to avoid any damage to the drone, but could later be expanded to its rotational movements as well. A saturation in speed, at 1.92 m/s, was thus implemented. This value remains in a range where aerodynamic drag should not have an impact on the course of the drone [7]. Considering all those aspects, the drone's inertia can almost be neglected. It is also possible to assume commanded inputs to be achieved immediatly [7]. All of those approximations will add more uncertainties to the model, but all of them will later be handled thanks to interval analysis. In the end, our model will be rather simplistic and so far would appear like this:

$$\begin{cases} \dot{x} = u_x \\ \dot{y} = u_y \ \ or \ \dot{\mathbf{w}} = \mathbf{u} \\ \dot{z} = u_z \end{cases} \qquad (2)$$

where $\dot{\mathbf{w}}$ corresponds to a velocity, and $\mathbf{u}$ to a command.

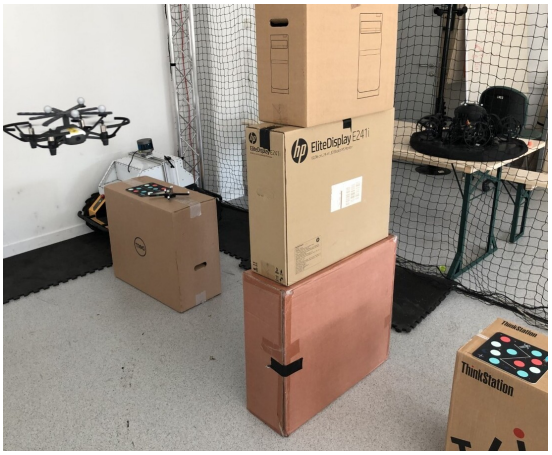

Fig. 1. **The flight environment and the drone**

B. The flight environment

The area in which the drone flies can contain several obstacles. Because of the way it was implemented the obstacles must be rectangular and parallel to the orthogonal directions of the spatial system. In the code, the environment has a set of boundaries and a list of rectangular obstacles, which can be inflated at will. A necessary inflation is to inflate the obstacles with a length equal to the longest diagonal of the drone (121.68 mm), so only a point is needed as a representation of the drone. The variety of obstacles can obviously be extended, as long as a function verifying potential collisions between a trajectory and the new type of obstacle exists. The area also contains several infrared sensors which can detect the drone with an accuracy down to around 2 mm, and with a detecting rate up to a 100Hz. The control frequency of the drone is thus chosen accordingly, at around 100 Hz as well. As the drone is placed in a closed environment, the installation of infrared sensors was natural. However, the monitoring of a system could be extended to outdoor complex systems, for instance with an interval observer using the available sensors [6], [11].

III. PATH PLANNING AND COMMAND OF THE DRONE

In this section, the path planning of the drone will be discussed. It is important to differentiate the global path planner, used to find a list of way points from the starting position of the drone to the destination, and the local command which generates the right commands so the drone can succesively attain all the way points.

A. Path planning

The path is determined by using an algorithm, which was strongly inspired by the Rapidly-exploring Random Trees *RRT* and its upgraded version *RRT** [9], [10] . The main difference between the implemented versions and the original ones lies in the fact that the dynamic model is not taken into account for the local planning motion. In the case of the drone, this approximation can be justified as the drone can locally move in all the directions in space without much constraints, unlike a car which requires a lot more manoeuvres to park for instance. The drone is a holonomic system.

Algorithm 1 RRT [9]

$V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$;
for $i \leftarrow 1, n$ **do**
  $x_{rand} \leftarrow SampleFree_i$;
  $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand})$;
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$;
  if $ObstacleFree(x_{nearest}, x_{new})$ **then**
    $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$;
  end if
end for
return $G = (V, E)$;

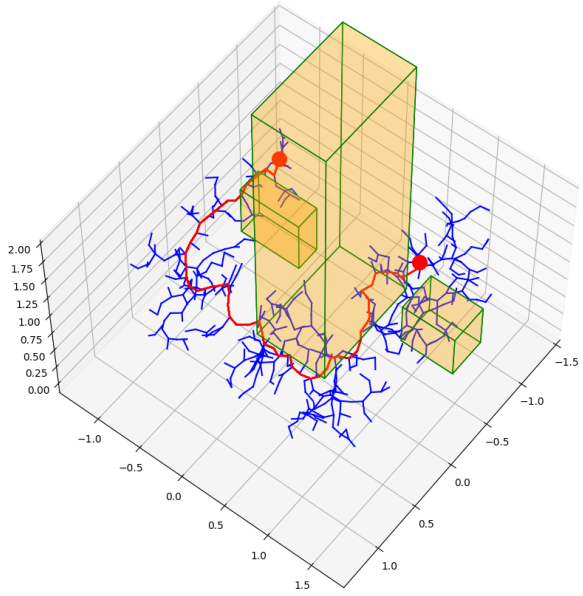

Fig. 2. **Expanding tree using *RRT* algorithm** between a starting point and a destination, with 3 obstacles in the scene

The principle of the RRT (**Algorithm** 1 [9]) is to generate an expanding tree of way points from a root (the starting position) with random nodes. The tree keeps expanding until it reaches the destination. At each step, a random position is drawn x_{rand} , then the nearest node belonging to the tree is determined. The new node introduced to the tree x_{new}, corresponds to the node at a step distance from $x_{nearest}$ in the direction of x_{rand} and free of collision. The *RRT* was first implemented because it was easier to. However it is proven and can easily be observed even with a relatively high number of iterations that this algorithm is not asymptotically optimal [9].

Let's discuss a few notations from the *RRT** algorithm (**Algorithm** 2 [9]) to understand it. In this case, the *Line* refers to the straight line path between its two arguments. The $Parent : V \to V$ function returns the parent node of its argument, the parent node of the root being the root itself (or a null pointer to identify it). The $Cost : V \to \mathbb{R}_+$ function returns the cost of the unique path between the argument and the root [9] (It measures a distance in our case). The c function returns the distance along a line. γ_{RRT*} refers to the search radius of neighbouring nodes, and η refers to the step length between a new node and its parent node. The **Algorithm** 1 and the **Algorithm** 2 begin the same way. However, before inserting a new node x_{new} to the tree, the *RRT** algorithm finds all the neighbouring nodes in a search radius and determines x_{new}'s parent optimizing the cost function (The First For loop of the algorithm). And then, the *RRT** algorithm considers x_{new} as a potential new parent for its neighbouring nodes, and rewires the tree to optimize the cost function as well (The Second For loop of the algorithm). *RRT** was proven to be asymptotically optimal, and was thus implemented in the final version.

Algorithm 2 RRT* [9]

$V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$;
for $i \leftarrow 1,n$ **do**
  $x_{rand} \leftarrow SampleFree_i$;
  $x_{nearest} \leftarrow Nearest(G=(V,E),x_{rand})$;
  $x_{new} \leftarrow Steer(x_{nearest},x_{rand})$;
  if $ObstacleFree(x_{nearest},x_{new})$ **then**
    $X_{near} \leftarrow Near(G=(V,E),x_{new},$
    $min\{\gamma_{RRT*}(log(card(V))/card(V))^{1/d},\eta\})$;
    $V \leftarrow V \cap \{x_{new}\}$;
    $x_{min} \leftarrow x_{init}$; $c_{min} \leftarrow \infty$;
    for $x_{near} \in X_{near}$ **do** //First For loop
      if $CollisionFree(x_{near},x_{new})$ & $Cost(x_{near}) + c(Line(x_{near},x_{new})) < c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$;
        $c_{min} \leftarrow Cost(x_{near}) + c(Line(x_{near},x_{new}))$;
      end if
    end for
    $E \leftarrow E \cup \{(x_{min},x_{new})\}$;
    for $x_{near} \in X_{near}$ **do** //Second For loop
      if $CollisionFree(x_{near},x_{new})$ & $Cost(x_{new}) + c(Line(x_{near},x_{new})) < Cost(x_{near})$ **then**
        $x_{parent} \leftarrow Parent(x_{near})$;
        $E \leftarrow (E \setminus \{(x_{parent},x_{near}) \cup \{(x_{new},x_{near})\}$;
      end if
    end for
  end if
end for
return $G=(V,E)$;

One of the advantage of the *RRT* algorithms is their time complexity which is following an ***O****(nlogn)* complexity in terms of processing, and an ***O****(n)* complexity in terms of queries [9]. There are other variants of the RRT algorithms, for the one implemented in the drone, there is a little bias which chooses directly the destination node, as the 'new random' node, at a specified rate. This variant is referred as Artificial Potential Fields (APF) [10].

B. Control of the drone

Thanks to the previous subsection, it is possible to obtain a set of several way points between a starting point and a destination. Since a clear path is planned, with the use of the infrared sensors in the area of flight, it was possible to have a closed loop system. Once again, the monitoring of trajectories could be further extended in cases where an interval observer is required [6]. In the case of an interval observer, the uncertainties linked to the observer would be greater and would need to be considered in our model based on interval analysis.

A PI (Proportional Integral) controller was implemented for the drone. This controller came naturally, since the coefficients could have been tuned thanks to interval analysis as well [13]. Besides, it is worth mentionning that the *RRT** algorithm can produce relatively long straight lines, as it rewires and find shortcuts between nodes. In the case of a Proportional controller, the drone could accumulate a lot of inertia even with a saturation. That's why in the case of a long distance between two way points, intermediate points were introduced. Any path folowing algorithms can be used to perform this task, even with wind[18] or for a vision-based control [19].

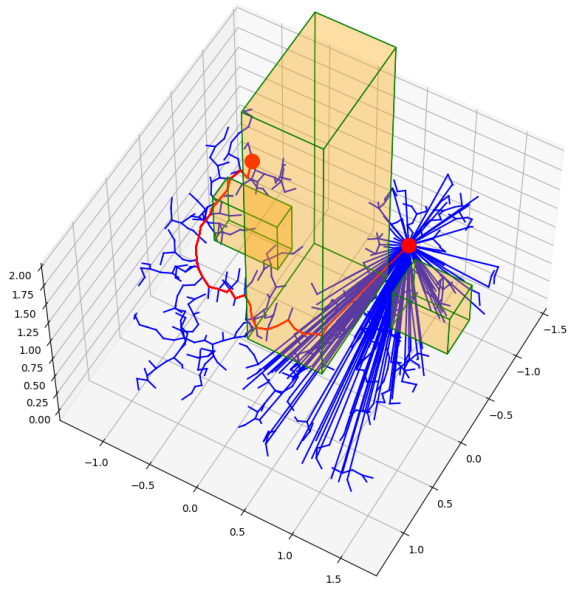

Fig. 3. **Expanding tree using *RRT** algorithm**

Obviously, each geometrical position of the way points cannot be reached perfectly by a drone. A margin was implemented so that a way point can be considered reached more easily. This margin was reduced to a minimum during the landing phase with the purpose of having the most accurate landing possible. Those margins add uncertainties to our model as well.

Considering the PI controller, it is possible to define more precisely the commanded inputs:

$$\begin{cases} u_x = K_P * (x_{waypoint} - x_{position}(t_n)) \\ \qquad + \ K_I * \sum_{i=n_{prev}}^{n} (x_{waypoint} - x_{position}(t_i)) \\ u_y = K_P * (y_{waypoint} - y_{position}(t_n)) \\ \qquad + \ K_I * \sum_{i=n_{prev}}^{n} (y_{waypoint} - y_{position}(t_i)) \\ u_z = K_P * (z_{waypoint} - z_{position}(t_n)) \\ \qquad + \ K_I * \sum_{i=n_{prev}}^{n} (z_{waypoint} - z_{position}(t_i)) \end{cases} \tag{3}$$

where n corresponds to the current index of time, and n_{prev} the previous index of time upon reaching the previous way point.

It is necessary to note that the coefficients K_P and K_I have a unit in s^{-1}. Even with the accuracy of the sensors (around 2mm) and the PI controller, the drone will never perfectly follow the path defined globally, there will always be uncertainties.

IV. APPLIED INTERVAL ANALYSIS AND OUR MODEL

A. A small introduction to interval analysis

The main underlying tool, used in our design methodology, is *interval analysis* [1]. There is a need to clarify some notations that will be reccurently used. The following notation $[x] \in \mathbb{IR}$ represents an interval $[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leqslant x \leqslant \overline{x}\}$. And $\mathbb{IR}$ is the set of intervals with real bounds. By abuse of notation, $[x]$ will also denote a vector of intervals, *i.e.*, a Cartesian product of intervals, a.k.a. a *box*. In our case, the boxes will obviously be three dimensionnals.

In this article, interval analysis is meant to take into account all the uncertainties. So the uncertainties will be considered as uniform distributions. Indeed, a uniform distribution allows to encompass even the worst possible cases [5]. Without going in depth, the notions of contractors and solvers will also implicitely be used in the next section [2].

B. Establishing our model based on the control

Thanks to the path planning section, and more particularly the local command in speed, a rather simplistic model of the drone behaviour was established:

$$\begin{cases} \dot{x} = u_x \\ \dot{y} = u_y \ \ or \ \dot{\mathbf{w}} = \mathbf{u} \\ \dot{z} = u_z \end{cases} \tag{4}$$

The command $\mathbf{u}$ was detailled in the previous section as well. The infrared sensors allow the user to have the position of the drone at a given time. Thus, it is an interval initial value problem, with an ordinary differential equation in the following form.

$$\dot{y} = f(y) \text{ with } y(0) \in [y_0] \text{ and } t \in [0, t_{horizon}] \tag{5}$$

It is assumed that the function $f : \mathbb{R}^3 \to \mathbb{R}^3$ is globally Lipschitz in y, and here corresponds to the flow. y would be the position of the drone, and $[y_0]$ corresponds to the uncertainty due to the sensors' accuracy. $t_{horizon}$ will be more detailed in a latter section, but it reprensents the time during which each simulation is made [3].

An arsenal of validated Runge-Kutta methods exists to compute the sets of boxes, solution to these types of ordinary differential equation [3]. A little reminder of what a validated numerical integration does at each step [3]:

Phase 1 One computes an *a priori* enclosure $[\tilde{y_j}]$ of the solution such that

- $y(t;[y_j])$ is guaranteed to exist for all $t \in [t_j, t_{j+1}]$, *i.e* along the current step, and for all $y_j \in [y_j]$
- $y(t;[y_j]) \subseteq [\tilde{y_j}]$ for all $t \in [t_j, t_{j+1}]$
- the step-size $h_j = t_{j+1} - t_j > 0$ is as large as possible in terms of accuracy and existence proof

Phase 2 One computes a tighter enclosure of $[y_{j+1}]$ at time t_{j+1}, such that $y(t_{j+1}, [y_j]) \subseteq [y_{j+1}]$.

In this article, we have mentionned the presence of several uncertainties which can impact the commanded inputs. That's why, an affine form for the commanded inputs was introduced in the model. An affine form provides a simple model which helps to have faster computation times for the monitoring, and encompasses largely the uncertainties.

$$\begin{cases} \dot{x} = [\alpha_x] * u_x + [\beta_x] \\ \dot{y} = [\alpha_y] * u_y + [\beta_y] \ \ or \ \dot{\mathbf{w}} = [\alpha] * \mathbf{u} + [\beta] \\ \dot{z} = [\alpha_z] * u_z + [\beta_z] \end{cases} \tag{6}$$

A model established with this form is a way to consider a large panel of uncertainties.

The presence of $[\alpha]$:

- corrects the assumption of immediate commanded inputs and includes the drone's speed profile
- takes into account all uncertainties linked to forces which have dependecies with the velocity
- takes into consideration uncertainties linked to the tuning of the coefficients K_P and K_I [13]

The presence of $[\beta]$:

- corrects all uncertainties linked to punctual pertubations
- considers ambient noises in the command and induced by the drone itself

Let's introduce scores of how well the model was built when the various uncertainties were considered. It is possible to introduce a coefficient $c_\cup$ measuring the number of times the computed boxes were intersecting with the trajectory, and another coefficient $c_\subset$ measuring the number of times the trajectory was actually included in the computed boxes. In this case, the trajectory are boxes with the length of the sensors' uncertainty.

$$\begin{cases} c_\cap = \frac{\text{Number of intersections}}{\text{Total number of comparisons}} \\ c_\subset = \frac{\text{Number of inclusions}}{\text{Total number of comparisons}} \end{cases} \tag{7}$$

The interval coefficients $[\alpha]$ and $[\beta]$ were determined for a collision free trajectory, and based on optimizing these two scores:

$$\left\{ [\alpha] * \mathbf{u} + [\beta] = \begin{pmatrix} [0,1] \\ [-0.4, 0.4] \\ [-0.4, 0.4] \end{pmatrix} * \mathbf{u} + \begin{pmatrix} [-0.1, 0.1] \\ [-0.1, 0.1] \\ [-0.1, 0.1] \end{pmatrix} \right. \tag{8}$$

V. MONITORING OUR TRAJECTORY WITH DYNIBEX

A. Principle of the monitoring

Now that a model has been established, it will be possible to compute a set of boxes representing the drone's behaviour. The trajectory of the drone should be inside those boxes. Those boxes will be computed thanks to the library DynIbex. Indeed, DynIbex will be used to resolve

the equations representing our model. DynIbex uses the theory of applied interval analysis, and affine arithmetic to solve various types of differential equations. In this case, the differential equation is rather simple, but DynIbex allows a relatively fast and validated integration. The Runge-Kutta method used in this article is Heun's method. With the help of the computed boxes, it will be possible to monitor the trajectory of the drone. Indeed since our obstacles are rectangular and in the same directions as the spatial system, they can easily be converted to boxes. DynIbex has a feature that allows us to check if boxes intersect or are subsets of each other. If we were to later introduce more complex obstacles, an interesting solution would be to decompose the complex obstacles into several rectangular obstacles.

The principle of the monitoring is then easier to implement. As the drone determines its trajectory, DynIbex will compute whether or not the drone will be within a set of boxes, guaranteed to reach the way points, and not collide with any of the obstacles. A similar work has been done, but the idea was not to monitor the trajectory with DyniBEX, but directly compute the trajectory of a system as a set of boxes, using an RRT algorithm [12]. The issue with this method in our case, is that the computed boxes would rapidly diverge unless the model was more accurate. Besides, it is likely that building an RRT tree, with boxes as nodes in a 3D space, might take a long time.

In terms of implementation, if the trajectory determined by the global planner were monitored by our solver and deemed incorrect because of a potential collision, the idea would be to inflate even more the obstacles with the purpose of adding a margin of security against collisions. So far, the monitoring is done, *a posteriori*, over the course of the whole trajectory. But with a little bit more time and implementation, it would be possible to imagine a real-time monitoring, the trajectory would be monitored continuously over a sliding horizon of time [17]. The monitoring would be focused on detecting potential collisions, induced by moving obstacles or by an innacurate trajectory, and would launch the planning of a new trajectory.

B. Observations and Results

In the previous section, a $t_{horizon}$ was mentionned, this $t_{horizon}$ represents the time during which each computation of the boxes should be made, before another measure of position is detected. Depending on this $t_{horizon}$, our model will have to be more refined with its uncertainties ($[\alpha]$ and $[\beta]$). Let's dicuss it with examples:

As it is visually observable in Figure 4 and Figure 5, when the uncertainties considered for $t_{horizon} = 1$ s were kept for a computation with $t_{horizon} = 3$ s, the computed boxes started to diverge. Indeed, the uncertainties considered were too rough, and since the computation time is greater, the computed boxes have the time to accumulate the uncertainties until this horizon of time. If we were to monitor this trajectory beforehand, we would definitively not let the drone fly as the computed boxes would detect a collision with the obstacles.

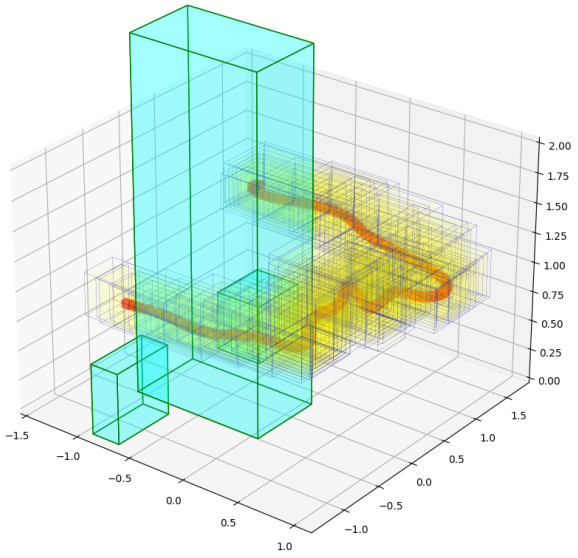

Fig. 4. **In yellow the computed boxes for $t_{horizon} = 1s$ with the adjusted $[\alpha]$ and $[\beta]$ (Collision free), in red the actual trajectory followed by the drone**

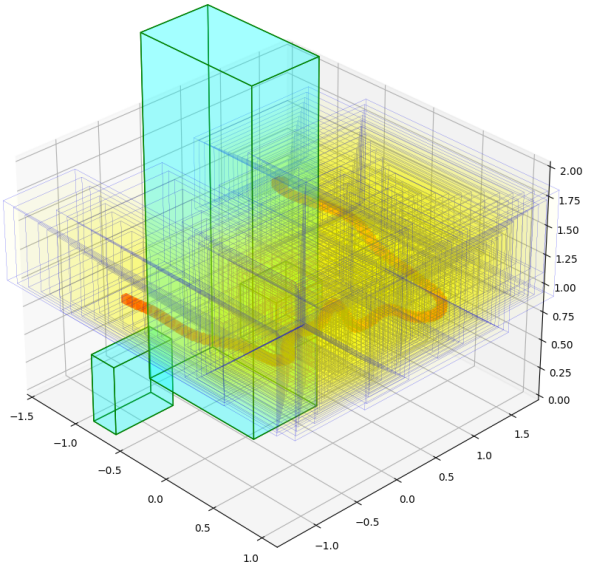

Fig. 5. **In yellow the computed boxes for $t_{horizon} = 3$ s with the adjusted uncertainties for $t_{horizon} = 1s$ (Collision), in red the actual trajectory followed by the drone**

However, since the model was built too simplistically, it is difficult to refine the uncertainties so that it could perfectly coincide with the reality and have a $t_{horizon}$ tending toward infinity.

As it can be seen in the Figure 6, although the computed boxes follow correctly the trajectory in the beginning, when the horizon of time is too far ahead, the simulation still starts to diverge and does not encapsulate the trajectory properly.

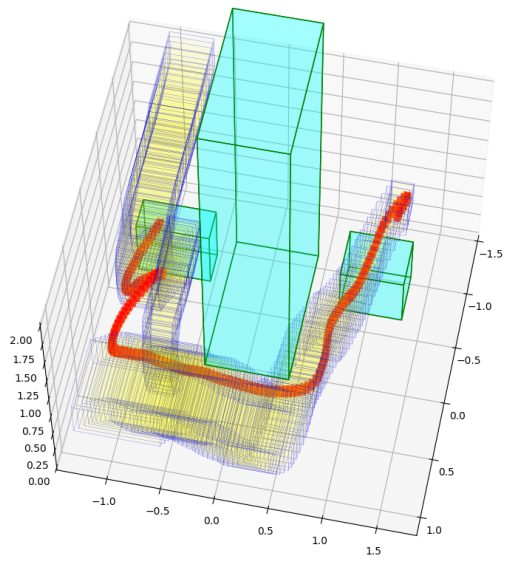

Fig. 6. **In yellow the computed boxes for $t_{horizon} = 15$ s with the adjusted uncertainties for $t_{horizon} = 1$ s, in red the actual trajectory followed by the drone**

Actually, the simulated trajectory is closed to how the drone would fly if it were in an open-loop control. Near the end of the trajectory, there are several tubes of computed boxes. As the flight lasted longer than $15s$, the several simulations happened to be visible. The effect is not in the Figures 4 and 5, because the various tubes overlap as their horizons of time are short.

There is a strong link between the creation of the model and the efficiency of the method. The scores using the adjusted interval coefficients $[\alpha]$ and $[\beta]$ for $t_{horizon} = 1$ s, with the following $t_{horizon}$, is a reflect of this link.

Scores and collision				
$t_{horizon}$	1 s	3 s	12s	15 s
$c_{\cap}$	1	1	1	0.786
$c_{\subset}$	0.687	0.676	0.899	0.59
collision ?	No	Yes	Yes	Yes

Naturally as $t_{horizon}$ increases, the uncertainties have more time to diverge (get bigger), and the scores $c_{\cap}$ and $c_{\subset}$ increase as they encompass a large part of the arena. However, the scores could also decrease for specific $t_{horizon}$, as the simulation would completely go off course in time. In the end, it is essential to tune the interval coefficients ($[\alpha]$ and $[\beta]$) and the security margins while ensuring a collision free trajectory, using the scores $c_{\cap}$ and $c_{\subset}$, but also while considering the right horizon of time depending on the system.

VI. CONCLUSIONS

The paper introduced a method to monitor trajectories using interval analysis. The method is based on balancing several aspects of the model. Indeed, for the method to work efficiently, there is a need to have a balance between the accuracy of the model, the uncertainties linked to the sensors and the observer, the inherent uncertainties induced by the system and the horizon of computation time. After finding the balance between all those criteria, the method will allow to determine the reliability of trajectories. As mentionned in the introduction, there is room for improvements on several aspects, and a case study with a more complex model and greater uncertainties could better demonstrate the efficiency of the method.

REFERENCES

[1] Moore, Ramon (1966). Interval Analysis. Prentice Hall, 1966.

[2] Jaulin, L., Kieffer, M., Didrit, O., & Walter, E. Applied interval analysis, 2001. ed: Springer, London.

[3] Alexandre dit Sandretto, J., & Chapoutot, A. (2016). Validated explicit and implicit Runge-Kutta methods. Reliable Computing electronic edition, 22.

[4] Rohou, S., Jaulin, L., Mihaylova, L., Le Bars, F., & Veres, S. M. (2017). Guaranteed computation of robot trajectories. Robotics and Autonomous Systems, 93, 76-84.

[5] Merlet, J. P. (2006). Interval analysis and reliability in robotics.

[6] Abadi, A., El Amraoui, A., Mekki, H., & Ramdani, N. (2020). Guaranteed trajectory tracking control based on interval observer for quadrotors. International Journal of Control, 93(11), 2743-2759.

[7] Kousik, S., Holmes, P., & Vasudevan, R. (2019). Technical Report: Safe, Aggressive Quadrotor Flight via Reachability-based Trajectory Design. arXiv preprint arXiv:1904.05728.

[8] Pepy, R., Kieffer, M., & Walter, E. (2009). Reliable robust path planning. International Journal of Applied Mathematics and Computer Science, 19(3), 413-424.

[9] Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7), 846-894.

[10] Pharpatara, P., Hriss, B., & Bestaoui, Y. (2016). 3-D trajectory planning of aerial vehicles using RRT. IEEE Transactions on Control Systems Technology, 25(3), 1116-1123.

[11] Thabet, R. E. H., Raissi, T., Combastel, C., Efimov, D., & Zolghadri, A. (2014). An effective method to interval observer design for time-varying systems. Automatica, 50(10), 2677-2684.

[12] Panchea, A. M., Chapoutot, A., & Filliat, D. (2017, December). Extended reliable robust motion planners. In 2017 IEEE 56th Annual Conference on Decision and Control (CDC) (pp. 1112-1117). IEEE

[13] Alexandre dit Sandretto, J., Chapoutot, A., & Mullier, O. (2015, November). Tuning PI controller in non-linear uncertain closed-loop systems with interval analysis. In 2nd International Workshop on Synthesis of Complex Parameters, OpenAccess Series in Informatics (Vol. 44, pp. 91-102).

[14] Frazzoli, E. (2001). Robust hybrid control for autonomous vehicle motion planning (Doctoral dissertation, Massachusetts Institute of Technology).

[15] Burlet, J., Aycard, O., & Fraichard, T. (2004, April). Robust motion planning using markov decision processes and quadtree decomposition. In IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 (Vol. 3, pp. 2820-2825). IEEE

[16] Lindemann, L., Pappas, G. J., & Dimarogonas, D. V. (2021). Reactive and risk-aware control for signal temporal logic. IEEE Transactions on Automatic Control.

[17] Alexandre dit Sandretto, J., Brendel, E., & Chapoutot, A. (2018). An Interval-based Sliding Horizon Motion Planning Method. IFAC-PapersOnLine, 51(16), 296-301

[18] Mcgee, T. G. et Hedrick, J. K.. Path planning and control for multiple point surveillance by an unmanned aircraft in wind. In : 2006 American Control Conference. IEEE, 2006. p. 6 pp.

[19] Courbon, J., Mezouar, Y., Gunard, N., & Martinet, P. (2010). Vision-based navigation of unmanned aerial vehicles. Control Engineering Practice, 18(7), 789-799.