Development of an Object-Oriented C++ Tool for Robust Model Predictive Control applied to UAV Path Tracking

Marcos Vinicius Teixeira Maciel, Marcos-vinicius.TEIXEIRA-MACIEL@student.isae-supaero.fr
Tutor: Caroline CHANEL, caroline.chanel@isae-supaero.fr

***Abstract*—This project focuses on leveraging probabilistic AI tools to tackle the research problem of designing Robust Model Predictive Controllers (MPCs) for UAV path tracking. While traditional MPCs exhibit inherent robustness, they struggle with handling various forms of uncertainties. To address this limitation, we translate the MPC problem into a probabilistic AI planning problem, specifically a Stochastic Shortest Path (SSP). Initially, Real-Time Dynamic Programming (RTDP) is implemented, followed by the adoption of a Tree Heuristic Tree Search (THTS) framework, utilizing the MaxUCT algorithm to combine Monte-Carlo and Full Bellman backups for optimal empirical action recommendations. Preliminary benchmarking studies demonstrate strong resistance to white noise perturbations and random winds. Furthermore, evaluations involving different types of uncertainties validate the overall robustness of the proposed controller. Finally, the path-tracking capability of the controller is tested through 1D and 2D trajectoriestion. The validation of the model is based on high fidelity numerical.**

Probabilistic AI, Robust Model Predictive Control, UAV Path Tracking, Stochastic Shortest Path, Real-Time Dynamic Programming, Tree Heuristic Tree Search, MaxUCT algorithm, Uncertainty, White Noise Perturbations, Random Winds, Path-Tracking.

I. Introduction

THE advancement of microprocessor technology has led to a significant increase in computational resources available on drones. This development has paved the way for the utilization of online real-time optimization-based control strategies. One such strategy is Model Predictive Control (MPC), which aims to optimize a given cost by looking into the future and calculating the best action at the current time step. While MPC offers a certain level of robustness, there is still room for improvement as existing robust MPC algorithms often rely on simplifications and assumptions of linearized models, leading to potential model errors.

In contrast, a Markov Decision Process (MDP) provides a probabilistic AI planning framework where an agent learns to take actions in an environment to maximize expected rewards. MDPs readily incorporate uncertainties and perturbations through probability transitions, offering a flexible approach. By transforming an MPC problem into an MDP, it becomes a stochastic shortest path planning problem, offering new possibilities for enhancing robustness.

This research aims to explore the MDP framework to improve the robustness of MPC, specifically focusing on the problem of UAV path tracking. To ensure real-time implementation, a quick and anytime MDP algorithm is preferred. Real-Time Dynamic Programming (RTDP), based on asynchronous Dynamic Programming, and the Trial Heuristic Tree Search (THTS) framework, combining tree search with on-the-go probability computations, are both suitable algorithms that satisfy the anytime nature. The THTS algorithm is particularly emphasized in this study.

This report begins by discussing the state of the art in robust MPC and MDPs, providing an overview of relevant studies. The novelty of the research is then justified by highlighting the core problem. The conversion of an MPC control problem into an AI MDP problem is explained, followed by the implementation of the RTDP algorithm on a simplified drone model in the pitch and roll axes. The obtained results are thoroughly discussed. Building upon the findings of RTDP, the MaxUCT algorithm is implemented under the THTS framework. A benchmarking analysis is conducted to explore parametric dependencies and the algorithm's robustness to uncertainties, using the simplified drone model. Finally, simulations of 1D and 2D reference trajectories are presented to demonstrate the capabilities of the proposed approach.

II. Project Definition

A. Content and key issues

Model predictive control (MPC) is an approach to controller design that involves on-line (onboard) optimization calculations. The online optimization problem takes account of system dynamics, constraints and control objectives. The fact that MPCs are inherently robust is an added advantage. However, there is a scope to improve the MPC's performance when uncertainties are involved. For this reason, a new approach of using AI Planning techniques like Markov decision process (MDP) as a MPC solver is worth looking into.

MDPs are a probabilistic approach applied to a control problem which can seamlessly handle uncertainties, thus making it more robust. Indeed, the main focus is to study and implement AI planning methods to improve the robustness of MPCs. In other words, we develop an MPC solver using the MDP framework.

The goal in a Markov decision process is to find a good "policy" for the decision maker i.e. the agent: a function $\pi(s)$ that specifies the optimal action that the decision maker will choose when in state "s". The objective is to choose a policy

$\pi^*(s,t) = \max_{\pi(s)}$ that will maximize some cumulative function of the future rewards, typically the expected discounted sum over a potentially infinite horizon. As the implementation for MPC purposes in real time is considered, a finite sliding horizon MDP is considered. A major difference between MPC and MDP is the cost function optimization. Model predictive control solves an optimization problem at each control interval through known numerical methods (Quadratic Programming for example). On the other hand, the optimization in the MDP problem is done by repeatedly iterating the values and policy of the states using Dynamic Programming.

MPC's come with their limitations; demanding hardware requirements, need for high computational power and the need to linearize/simplify the nonlinear model before the optimization process. A challenge in MDPs is obtaining the transition probabilities, which plays a significant role in modelling uncertainties and disturbances.

B. State of the Art

Model Predictive Control (MPC) represents a closed-loop optimal control problem with a receding horizon. [1]. MPCs work very well in optimization problems that contain strong-constraints[2]. Various algorithms to implement MPCs such as Dynamic Matrix Control (DMC), Model Algorithmic Control (MAC) and Internal Model Control (IMC) have been extensively studied [2]. MPCs were initially implemented in the process industry due to the slow nature of the systems involved which allowed time for MPC computation [3]. However, the development of stronger microprocessors for computation has heralded the entry of MPC into on-board control problems such as drones through methods like parallel computation and GPU utilization[3]. By using lower order prediction models and simplified linear models for system identification, the computation time for MPCs can be greatly reduced so as to run on real time systems[4]. MPCs have been utilized extensively for online problems related to trajectory control and path-tracking[5]. Although study on theoretical feasibility and stability is difficult, it has been proven for highly nonlinear systems such as the autopilot systems of unmanned surface vehicles[6]. Furthermore, the stability of MPCs in the aircraft reference tracking problem has also been proved for simplified aircraft models[7]. Thus the transition of MPCs from slow systems and offline computation to faster systems with online computation has begun.

A crucial area of research in the MPC domain has been the study of robustness i.e the ability of the controller to manage uncertainties and disturbances in the system being controlled. It has been shown that while MPCs are not inherently more or less robust than classical feedback, its parameters can be much easily tuned for robustness[2]. Stochastic and robust MPCs have and are being studied extensively to test their ability to handle uncertainty and disturbances. Most current methods are not suited for industry and practical use except in highly constrained situations as they are too complex for implementation in the systems that we know about[8][9]. These Robust MPC techniques are highly conservative because most cases assume an uncertainty description that consists of bounds on the unknown parameters. Such techniques fail to take into account information that is often available about the distribution of uncertainty[10]. Furthermore, recursive feasibility and convergence properties have been achieved by assuming that the noise is bounded and by resorting to min-max algorithms and worst case scenarios. Conservative solutions arising from such methods lead to complex on-line and off-line MPC designs to properly estimate the effects of noise along the considered prediction horizon[11].

The stochastic MPC taking into account noise and uncertainties has been compared with model-based reinforcement learning methods such as differential and dynamic programming[7]. Dynamic programming additive cost approaches to optimal control have been found to easily extend to the stochastic optimization problem of the MPC. The problem of MPC path tracking and probabilistic AI algorithms can be brought under the general umbrella of the Stochastic Shortest Path problem (SSP)[12]. From the probabilistic planning context, Markov Decision Processes (MDP) is a framework in which an agent optimizes a policy – a function that maps state to actions – keeping into account possible future states and try to maximize/minimize the expected sum of rewards/costs in a given time horizon[12]. MDPs represent a probabilistic AI decision making framework for stochastic control processes[13]. Within the MDP framework, various algorithms to take into account probabilistic models and solve them such as PILCO have been developed[14].

However, these models search the entire space of states and actions which take a lot of computational resources and are time consuming.

MDP algorithms have been found to show promising results in several online decision making problems[15]. Most of these algorithms rely on a more constrained sampling method and heuristics in order to guide the system to the desired goal based on the most promising current policies[16]. One such strategy is real-time dynamic programming (RTDP) that does not have to evaluate the entire state space in order to develop a near optimal policy, and has a good anytime behavior[16]. It has been shown that the theory of asynchronous Dynamic Programming can be used to show that RTDP converges in probability to optimal solutions when applied to several types of real-time problem solving tasks involving uncertainty[17]. By implementing a cost based on the current state of the controlled system, a closed loop control policy can be implemented; this kind of feedback is closely associated with negative feedback in classical controls[18]. However, an undesirable effect of this strategy is that unlikely paths tend to be ignored[16]. Thus, a key issue that is to be addressed is the trade-off between short and long term performance of the agent i.e. the agent has to improve its long term performance even when this may require sacrificing its short-term performance[18]. In such situations, algorithms such as Labelled Real Time Dynamic Programming (LRTDP) can prove to be a strong prospect[16].

RTDP algorithms have been implemented in drones in real time in order to optimize journey time and energy consumption[19]. They have also been used for guidance in cluttered environments[3][20]. Interestingly, RTDP algorithms have been shown to be capable of being made more robust[21].

Thus, these algorithms not only show promise to run quicker with shorter computational time, but also to provide more flexible optimization cases and criterion.

MCTS Algorithms are an efficient class of algorithms that can handle the sequential decision making problems in a very simplified manner. These are efficient in handling large search spaces well due to selective sampling of promising actions[22]. Algorithms like ε -greedy tree search and Upper confidence tree search (UCT) can perform comparatively well in identifying a good action[23]. MCTS can take long-term rewards into account even with distant horizons[24]. This class of algorithms gained popularity after succeeding in solving the most difficult problem of computer Go, but has also proved beneficial in a range of other domains[25].

The two major steps in an MCTS algorithm are: Approximation of the value of an action can be obtained using simulations; and so obtained values of all the promising actions would lead to the better refined policy and hence convergence[25]. The Quality of the Monte Carlo simulations can be enhanced by choosing the actions based on cumulative reward, which can be obtained by tracking the visited states in the generated tree structure[25]. The fact that intermediate states can be ignored when evaluating saves a significant amount of computational time and effort, making the MCTS algorithm more suitable for solving problems online[25].

Directing the search towards promising areas of the search tree and thereby obtaining good anytime behavior is one of the key functionalities of the UCT algorithm. Updating the backup function based on the best outcome of the sampling instead of aggregated outcome has led to a new variant called Max UCT algorithm[26]. MaxUCT algorithms combine Monte-Carlo and Full Bellman backups by merging action-value backup function of the previous case with the state-value backup function of the latter. The action selection is based on the balanced exploration and exploitation[26]. Trial-based Heuristic Tree Search (THTS), that subsumes these approaches and distinguishes them based on five ingredients: heuristic function,backup function, action selection, outcome selection, and trial length[26]. The THTS schema bridges the gap between Dynamic Programming, MCTS, and Heuristic Search algorithms for finite-horizon MDPs[26]. THTS algorithm is exclusively for finite-horizon MDPs and completely works on the fact that the search space is a tree structure[26].

As the search space of the problem increases, the high stochastic branching factors are a matter of concern where the performance of these algorithms is degraded. In those cases, state aggregation or abstraction is used as a method of reducing the stochastic branching factor for MCTS[27]. It is seen as a popular technique to address the search space scaling issues in AI problems and operations research. The idea is to work with an abstract state space which is basically a group of the ground states. By treating these groups of states as a unit and by ignoring irrelevant state information, the process of finding an optimal solution is much quicker[28]. The coarser the abstraction the larger the reduction in the state space and vice-versa, which will have a direct impact on the efficiency of the solving the problem[28]. However, there must be a trade-off between minimizing information loss and maximizing state space reduction when selecting abstractions[28]. Similar to the idea of state aggregation, an algorithm which proposes to reduces the search space by dynamically grouping together states with similar cost-to-go values is studied[29]. This schema is termed as pre-specified aggregations and is generally infeasible unless in the MDPs are already solved[29].

C. Essential definitions

1) Markov Decision Process: Consider a discrete system in which all states are available (observable) at any given point in time. Then, a Markov Decision Process is a tuple (S, A, D, T, R) where:

- S is the finite set of all possible states of the system, i.e., the state space.
- A is the set of all actions.
- D represents the time steps at which actions should be taken.
- $T : S \times A \times S \times D \rightarrow [0, 1]$ is a transition function that maps the probability $T(s_1, a, s_2, t)$ of transitioning to state s_2 if action a is executed in state s_1 at time t.
- $R : S \times A \times S \times D \rightarrow \mathbb{R}$ is a reward function that assigns a reward $R(s_1, a, s_2, t)$ for transitioning to state s_2 if action a is executed in state s_1 at time t.

2) Stochastic Shortest Path MDP: A Stochastic Shortest Path (SSP) Markov Decision Process (MDP) is a tuple (S, A, T, C, G) where:

- S, A, and T have been explained previously.
- $C : S \times A \times S \rightarrow [0, +\infty)$ is a cost function that assigns strictly positive costs.
- $G \subseteq S$ is the set of all goal states.

A condition for the the SSP is that there should exist at least one proper policy.

3) Value Function for a SSP MDP: The value of a state s under a policy π is the total expected cost, starting in s before it reaches a goal $g \in G$:

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_\pi(s_t, s_{t+1})\right] \tag{1}$$

where γ is the discount factor, and $\gamma = 1$ for a finite-horizon MDP.

4) Q Value under a Value Function: The Q value of a state-action pair (s, a) under a Value function V is the one-step look-ahead computation of the value of choosing action a in state s:

$$Q(s, a) = \sum_{s' \in S} T(s, a, s') \left[C(s, a, s') + V(s')\right] \tag{2}$$

5) Optimal Value and Policy: The optimal value of a state s is given as:

$$V^*(s) = \begin{cases} 0 & \text{if } s \in G \\ \min_{a \in A} Q^*(s, a) & \text{if } s \notin G \end{cases} \tag{3}$$

where the optimal Q value is given as:

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s')[C(s, a, s') + V^*(s')] \tag{4}$$

Therefore, the optimal policy $\pi^*(s)$ is defined as:

$$\pi^*(s) = \arg\min_{a \in A} \sum_{s' \in S} T(s,a,s')[C(s,a,s') + V^*(s')] \quad (5)$$

D. Justification of the potential degree of novelty

While studies on Robust Model Predictive Controllers (MPC) have been conducted previously, an approach based on probabilistic AI planning methods has not been explored to our knowledge. Implementations of model-based reinforcement learning algorithms have mainly targeted fuel optimization and obstacle/collision avoidance systems. Our research aims to bridge the gap between Control Theory and Artificial Intelligence by integrating Model Predictive Control methods with Markov Decision Processes (MDP).

Interestingly, both MPC and MDP represent similar classes of stochastic optimization problems, allowing for an analogy between these concepts as illustrated in Figure 1.

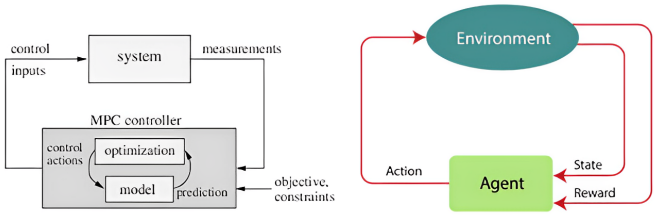

Fig. 1. Comparison of MPC and MDP Schemes

This comparison highlights the correspondence between the terminologies of MPC and MDPs. By representing control optimization problems as Markov Decision Processes, the controller is replaced by an agent that learns an optimal policy based on the environment. Both concepts involve states to represent the current system condition. While MPC optimization is guided by objectives and constraints, MDPs are driven by rewards, creating a parallel representation of the same problem.

A notable advantage of MDP representation is the avoidance of linearizing nonlinear systems, a step common in MPC optimization. In contrast, MDPs can compute transition probabilities regardless of the model type, significantly reducing model errors.

Another relationship between MPC and MDPs is the concept of a finite horizon. Calculating a limited number of future states to determine control inputs aligns with the idea of prediction horizons in MPC. This concept can be adapted in MDP algorithms as a finite-horizon MDP, thereby limiting the scope of future states and actions.

The motivations behind this approach can be summarized:

- MDP algorithms are adaptable to both linear and nonlinear systems, accommodating more accurate models.
- Incorporating nonlinearities like saturations becomes seamless in the action space of MDPs.
- Convergence guarantees for infinite-horizon MDPs extend to soft guarantees for finite-sliding horizon MDPs under certain conditions.
- MDPs are more versatile in addressing uncertainties, disturbances, and perturbations.
- Utilizing uncertainty distribution information is feasible within an MDP framework, unlike traditional MPC.
- Heuristics can drive the solution of the overall cost function in an MDP, enhancing its performance over a standard MPC solver.

These advantages position MDPs as strong contenders for MPC-like solvers to tackle control optimization challenges. The previous implementation in Python lacked the real-time capability for on-drone compilation. In contrast, our work has been implemented in C++ and optimized to enable online control. This advancement allows for real-time control and a more efficient execution on the drone.

E. Aims and Objectives

This research aims to bridge the gap between theoretical concepts and real-world applications by implementing and optimizing a theoretical framework for unmanned aerial vehicles (UAVs). The main objectives are as follows:

1) Translate Theory to Practical Reality: The primary goal is to translate the theoretical framework from prior research into practical application on UAVs. By encapsulating the finite-horizon Model Predictive Control (MPC) problem within a Stochastic Shortest Path Markov Decision Process (MDP) framework, we intend to seamlessly transition from theory to operational practice.

2) Efficient Object-Oriented C++ Implementation: This study replaces the prior Python implementation with an optimized Object-Oriented Programming (OOP) approach in C++. The new implementation aims to minimize compilation times, enhance runtime performance, and ensure code modularity and maintainability.

3) Real-Time Adaptive Decision-Making: With a focus on real-time applications, the OOP C++ implementation will enhance UAV decision-making in dynamic scenarios. Leveraging C++ and object-oriented design, the research seeks to empower UAVs with swift adaptability to changing conditions and inputs.

4) Empirical Validation via UAV Testing: Empirical validation of the OOP C++ implementation will be conducted to assess its practical effectiveness. The UAV's decision-making, responsiveness, and trajectory tracking precision will be tested in scenarios involving uncertainties and perturbations.

5) New State Representation through Relative Position: A novel grouping technique will be employed, utilizing relative position as the state. This approach aims to enhance the UAV's perception and decision-making by effectively capturing its position-based error, allowing for more intuitive and effective trajectory tracking.

6) Quantifiable Performance Metrics: The effectiveness of the OOP C++ implementation will be gauged through specific performance metrics, encompassing compilation time, computational efficiency, response latency, trajectory tracking precision, and system stability.

In conclusion, this research bridges theory and practice, optimizing UAV control through an OOP C++ implementation.

By leveraging the novel state representation and addressing challenges in real-time decision-making and trajectory tracking, the study strives to advance UAV autonomy, enabling precision and adaptability in complex environments.

III. Implementation of RTDP for the XY UAV Model

A. Dynamic modeling

For modeling the dynamic behavior of the drone the discretized state-space modeling was chosen. In equations 6 e 7 it's possible to observe the representation of states in the x and y directions.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -9.81 \end{bmatrix} \theta \tag{6}$$

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 \\ 9.81 \end{bmatrix} \phi \tag{7}$$

Where : θ and ϕ are the pitch and roll angles.

Nonetheless, in order to carry out the implementation, discretization is required. This requirement was specified as 20 ms, aligning with the response frequency of the drone's internal controller.

B. Transition probability using Monte Carlo Sampling

To acquire the transition probabilities necessary for performing the Value Update, a Monte Carlo Sampling technique was employed. This involved utilizing a simplified version of the drone model and applying a control input to it. The model was then exposed to input disruptions represented by Gaussian Noise, characterized by a mean of 0 and a standard deviation of 0.03. This standard deviation was deliberately set lower than the real control inputs that were administered. In this way, we have the discretized equation for the control of the drone, for example, the equation 8 represents the discretized y-axis.

C. Defining the Cost Function

The decision was made to incorporate the square error of both position and velocity into the cost function. However, it became evident that in order to achieve a satisfactory outcome, it was necessary to assign a weight to the velocity term. As a result, the cost function employed in the RTDP algorithm takes the following form:

$$R(s_t, s_{t+1}) = (x_{t+1} - x_{\text{goal}})^2 + \alpha(v_{t+1} - v_{\text{goal}})^2 \tag{8}$$

where:

- s_t and s_{t+1} denote the current state and the subsequent state of the drone, respectively.
- x_{t+1} signifies the position of the drone in the subsequent state.
- v_{t+1} represents the velocity of the drone in the subsequent state.
- x_{goal} represents the desired positional goal.
- v_{goal} represents the desired velocity goal.

The procedure for selecting the parameter α involved examining the optimal response within a single time unit (horizon = 1)for a uniform time span. The α parameter was varied across the range of 0 to 0.5. Figure 2 illustrates a graphical representation of this curve for certain values. The value of α chosen for subsequent simulations was 0.0035.

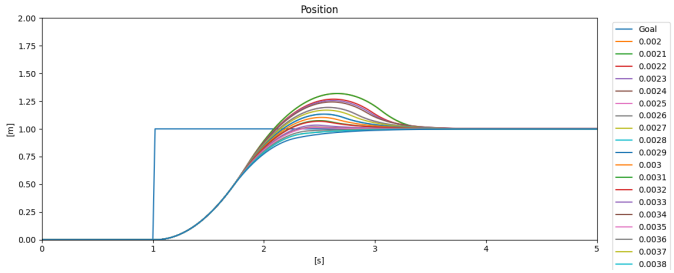

Fig. 2. Response for horizon equal to 1 for different α

D. RTDP algorithm for the drone

The pseudocode (Algorithm 1) that demonstrates the utilization of the RTDP algorithm for the drone.

RTDP algorithm is engaged in trial runs aimed at deriving a variety of trajectories within a defined prediction horizon. During this procedure, the values linked to the states undergo consistent updates. These RTDP trials are executed as long as there remains available time, signifying their continuation until the drone's sampling time has passed and the drone requires repositioning. This sequence of actions is reiterated ceaselessly until the objective is attained, thereby accomplishing the task of path tracking.

Algorithm 1: RTDP Algorithm for the Drone

```
Initialization of tolerances and parameters;
Assign Values V(s) to a Heuristic h(s);
while goal Not reached do
    while there is time do
        Update RTDP starting state to current drone
          state;
        while prediction horizon not reached do
            for all actions a do
                Monte Carlo Simulations - P(s'|s,a);
                Compute Q-Value - Q(s,a) =
                  ∑ P(s'|s,a) · (R(s,a) + γV(s'));
            end
            a ← arg min_{a∈A} Q(s,a);
            Update Value V(s) = min_{a∈A} Q(s,a);
        end
        Sample next state from distribution s';
        Apply action a and move to next state;
    end
end
```

E. Aggregations

1) relative states: At the project's inception, it was decided that leveraging previously computed states could enhance the code's performance. However, addressing non-stationary references presented a challenge. This stemmed from the requirement for a state to maintain a consistent cost function, thereby precluding the use of a fixed reference point. Consequently, a proposal emerged to employ the concept of error – namely, the disparity between the state and the reference point. The primary goal was to curtail the number of computed states (now termed as errors). This transformation is effectively depicted in Figure 3.

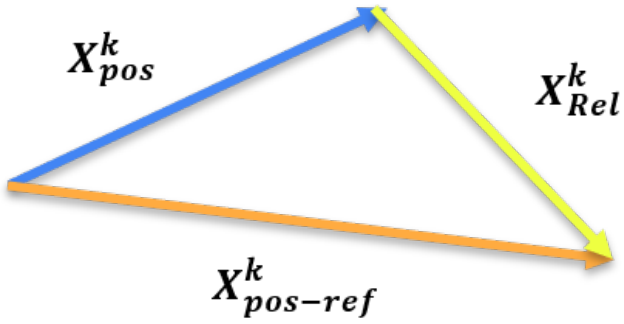

Fig. 3. Relative state representation

Following the discretization process, which entails replacing the static state in Equation 6, and taking into account that the reference position (goal) remains constant for each optimization, we arrive at Equation 9.

$$X^{k+1}_{rel} = (I - A_{dis})X^k_{pos-rel} + A_{dis}X^k_{Rel} - B_{dis}U^k \quad (9)$$

2) Discretization of states: The discretization of states, in this case the error, involves simply grouping all states within a particular region. This is of paramount significance as the frequency with which a state is accessed holds key relevance for probabilistic calculations. Without adopting this discretization approach, it is highly likely that certain states will not be accessed more than once. Figure 4 elucidates this concept.

3) Discretization of states: State reuse entails making use of previously calculated trees. This approach can result in a shift from a tree structure to a graph structure. In other words, when a state is reused, its entire lineage is also employed anew. This unchecked growth can potentially lead to the formation of a graph over time.

IV. Analysis and results

A. 1 D analysis y-axes

By analyzing figures 5, 6, and 7, it becomes evident that the response was quite satisfactory when a ramp or unit step input was utilized. However, when dealing with a sinusoidal reference input, even though convergence was achieved, there was an intriguing response that emerged. This atypical outcome can be linked to a constraint highlighted in equation 9. To better understand this constraint, it's essential to focus on a constant X_{Rel}. Our goal necessitates that an action U applied to a state X should lead to another fixed state, discounting any noise's impact (minimal influence on overall development). Nonetheless, it's important to note that the relative state term also plays a role in shaping the attainable state. Essentially, optimization loses its accuracy as the objective fluctuates. It's worth underscoring that solely the reference velocity is susceptible to this influence.

It's important to emphasize that despite this formulation error, the system converged; in other words, optimization managed to handle the uncertainty caused by relative velocity. Generally, the model can manage this uncertainty quite effectively when it pertains to staying within the reference. However, when it's attempting to enter the reference trajectory, it encounters a certain difficulty in managing this variation. In the figure above, it's possible to observe in the legend the average trial count plus or minus 2 times the standard deviation. This value represents how many paths were simulated during the optimization's feasible time. In this document, we won't delve extensively into this concept, as the reuse of precalculated states causes this value to differ from the number of children of this node in the tree, what happens when state reuse is not employed.

B. 2D analysis

For Figure 8, the simulation was conducted considering coupled x and y states. In other words, each feasible action consisted of a set of two angles. It's noticeable that as time progresses, the system improves its stability. This phenomenon occurs because the tree/graph becomes deeper with the advancement of the simulation.

In the second case, depicted in Figure 9, the states were decoupled. However, in order to allow for a different analysis from the 1D scenario, the precalculated states could be used in any direction. This means we have only one graph for both directions. A significant instability can be observed along the x-axis, while there's a certain random loss of balance along the y-axis (which initially started aligned with the reference). This instability characteristic likely arises due to the variability in depth (horizon) among nodes. Unfortunately, there wasn't sufficient time available to analyze or seek a solution for this issue. Implementing such a solution would be highly advantageous from a computational standpoint.

V. Conclusions

The initial goal of developing the C++ code was achieved. However, using the relative error as the key for the nodes introduced a constraint related to the variation in the reference object's velocity. This implies that this aggregation approach allows following or moving towards constant velocity objectives, and conditionally, towards objectives with velocity

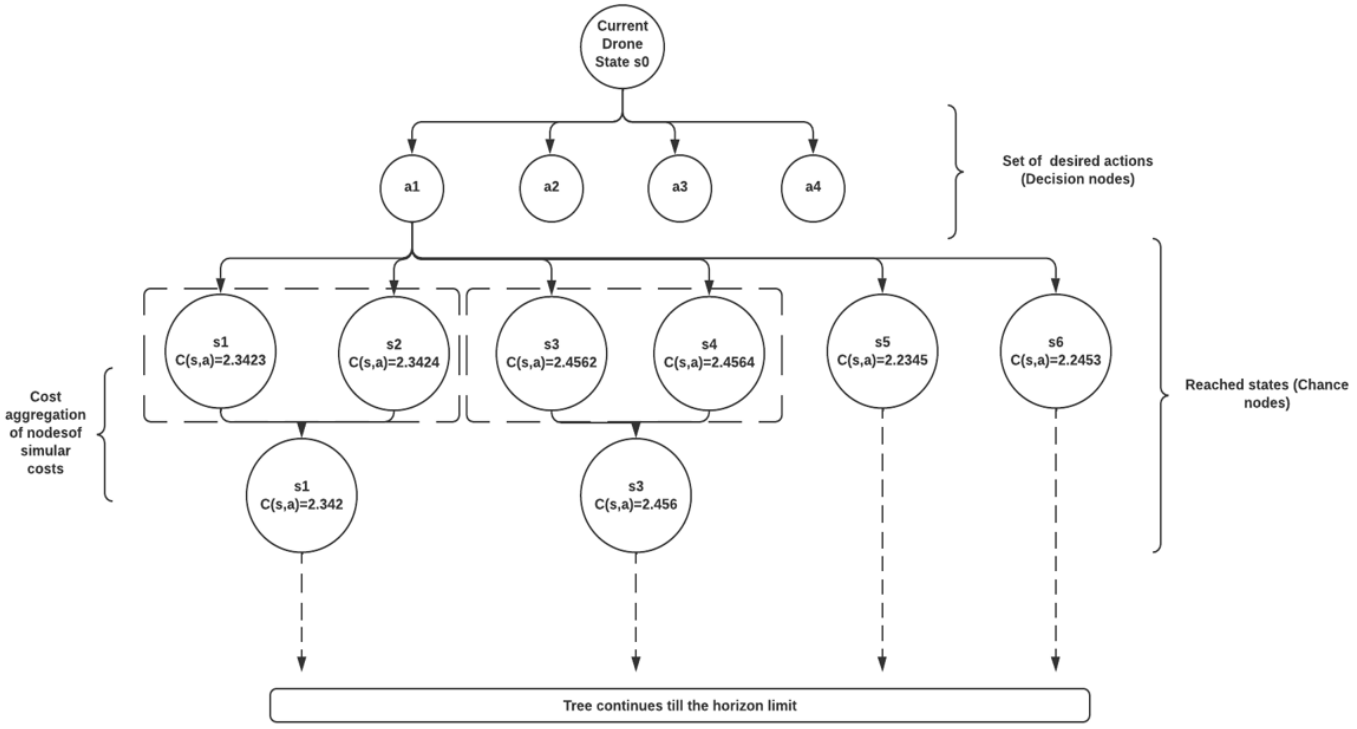

Fig. 4. Discretization representation

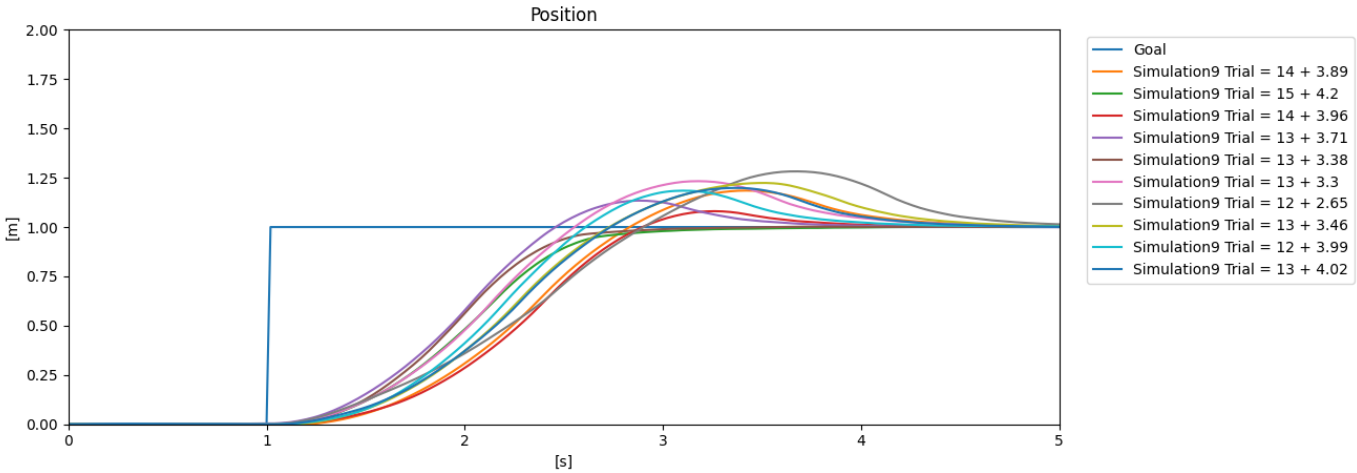

Fig. 5. Unitary step, horizon = 10, Possible actions = 4, Noise = 0.03

variations. If these variations are of low magnitude, the model can manage the error generated by them.

One possible solution to this issue would be to replace the term X^k_{Rel} with the term "$(I - A_{dis})X^k_{pos-rel} + A_{dis}X^k_{Rel}$" as the key for the node. However, this change would require developing a cost function for the term "$(I - A_{dis})X^k_{pos-rel} + A_{dis}X^k_{Rel}$". Implementing this solution would even enable considering future objectives instead of assuming a constant objective.

The reuse of pre-calculated trees enabled the model to create a kind of memory. This means that the control improved over time as the system accumulated experience. Nevertheless, when this information was transferred between different controllers, a sort of confusion arose, as observed in the last graph (Figure 9).

In conclusion, it is evident that the subject matter addressed in this scientific article holds a significant degree of novelty and presents considerable challenges. Despite the limitations imposed by the relatively short duration of the research period, the depth of exploration remains incomplete. Therefore, the potential for further investigation is substantial.

Outlined below are several avenues for future research that

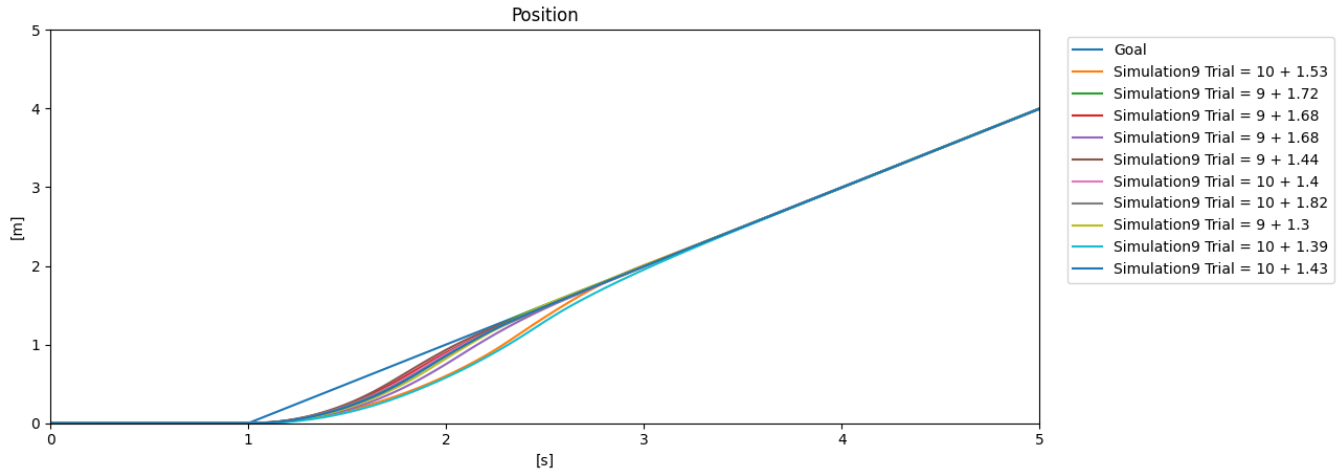

Fig. 6. Ramp , horizon = 10, Possible actions = 4, Noise = 0.03

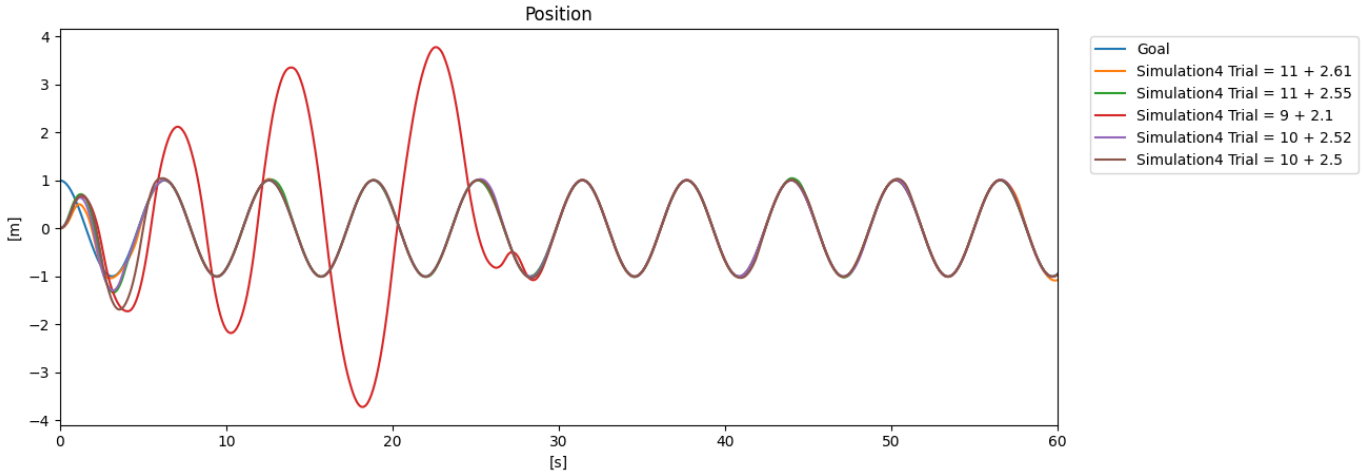

Fig. 7. Cosine, horizon = 10, Possible actions = 4, Noise = 0.03

could enhance our understanding of the topic:

- Study on the Use of State Restrictions: Delving deeper into the application of state restrictions could unveil valuable insights into their impact on the system's behavior and performance.
- Constant Restrictions and Dynamic Restrictions: A comparative analysis between constant and dynamic restrictions could shed light on the advantages and limitations of each approach, contributing to a more comprehensive understanding of their implications.
- Development of a Model Incorporating Node Speed into Key Calculation: The creation of a model that integrates node speed into the determination of key parameters holds promise for refining accuracy and adaptability, potentially leading to improved system performance.
- Analysis of the Influence of Reference Speed on Optimization: Investigating the influence of reference speed on the optimization process is essential to grasp the intricate interplay between dynamic variables and system efficiency.
- Real-Time Testing on UAVs with and without Disturbances: Conducting real-time testing on UAVs, both under normal conditions and in the presence of disturbances, is crucial to validate the robustness of the results obtained from simulations, thus establishing the credibility and applicability of the proposed methodologies.

In light of these potential avenues for future exploration, it is evident that the current study merely scratches the surface of a complex and promising field. By embarking on these suggested research directions, the scientific community can collectively contribute to the advancement of knowledge in this domain.

VI. References

[1] Zhu, B., Zuo, Z. and Ding, Z. (2020) 'Model Predictive Control for Discrete-time Linear Systems with Finite-time Convergence', Proceedings of the IEEE Conference on

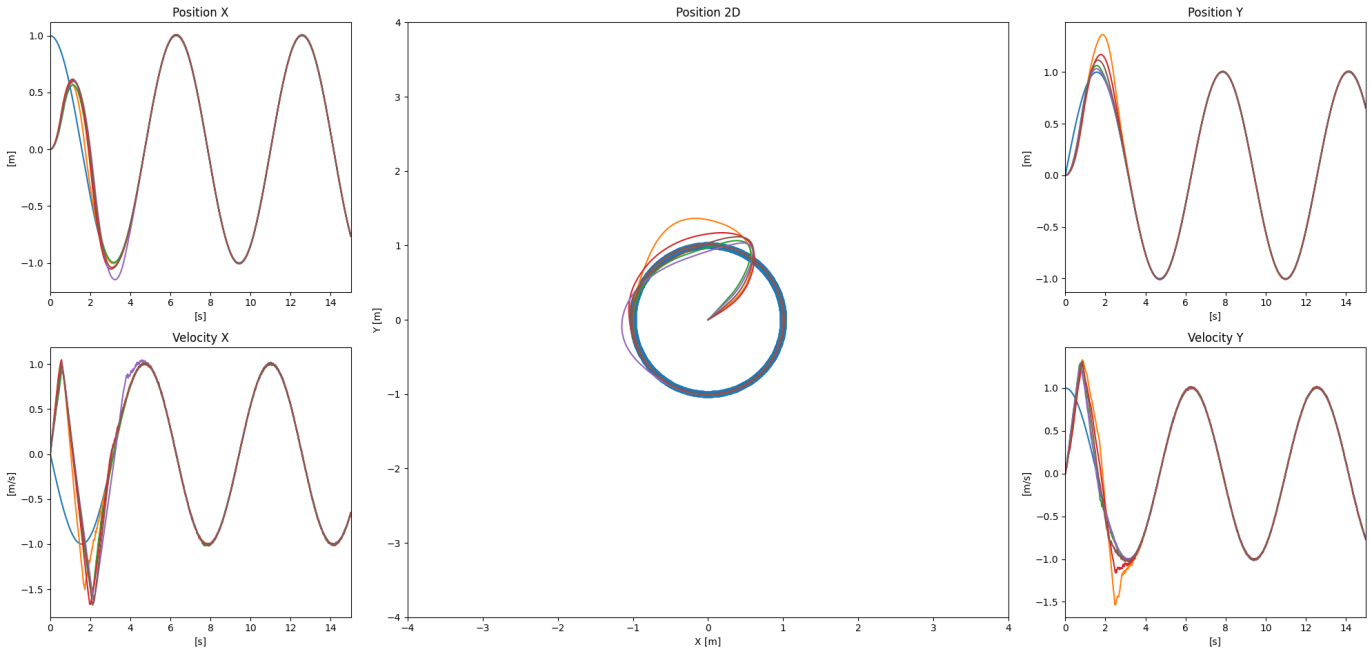

Fig. 8. 2D elliptic Coupled, horizon =2, PA = 4, Noise = 0.03, Pos. actions = 3

Decision and Control, 2020-December(Cdc), pp. 3531–3536. doi: 10.1109/CDC42340.2020.9303991.

[2] Morari, M., Garcia, C. E. and Prett, D. M. (1988) 'Model predictive control: Theory and practice', IFAC Proceedings Volumes, 21(4), pp. 1–12. doi: 10.1016/b978-0-08-035735-5.50006-1.

[3] Williams, G., Aldrich, A. and Theodorou, E. A. (2017) 'Model predictive path integral control: From theory to parallel computation', Journal of Guidance, Control, andDynamics, 40(2), pp. 344–357. doi: 10.2514/1.G001921.

[4] Takahama, T. and Akasaka, D. (2018) 'Model Predictive Control Approach to Design Practical Adaptive Cruise Control for traffic jam', International Journal of Automotive Engineering, 9(3), pp. 99–104. doi: 10.20485/jsaeijae.9.399. [5] Mahe, A., Pradalier, C. and Geist, M. (2018) 'Trajectory-control using deep system identication and model predictive control for drone control under uncertain load', 2018 22nd International Conference on System Theory, Control and Computing, ICSTCC 2018 - Proceedings, pp. 753–758. doi: 10.1109/ICSTCC.2018.8540719.

[6] Simon, D. (2014) Model Predictive Control in Flight Control Design: Stability and Reference Tracking. doi: 10.3384/lic.diva-103742.

[7] Annamalai, A. (2012) 'A review of model predictive control and closed loop system identification for design of an autopilot for uninhabited surface vehicles', Technical Report.

[8] Mayne, D. (2016) 'Robust and stochastic model predictive control: Are we going in the right direction?', Annual Reviews in Control, 41, pp. 184–192. doi: 10.1016/j.arcontrol.2016.04.006.

[9] Campo, P. J. and Morari, M. (1987) 'Robust Model Predictive Control.', Proceedings of the American Control Conference, pp. 1021–1026. doi: 10.1007/978-1-4471-5102-92-3.

[10] Cannon, M., Kouvaritakis, B. and Wu, X. (2008) Probabilistic constrained MPC for systems with multiplicative and additive stochastic uncertainty, IFAC Proceedings Volumes. IFAC. doi: 10.3182/20080706-5-kr-1001.02587.

[11] Farina, M. et al. (2014) 'Output feedback model predictive control: A probabilistic approach', IFAC Proceedings Volumes (IFAC-PapersOnline), 19, pp. 7461–7466. doi: 10.3182/20140824-6-za-1003.02283

[12] Kolobov, A. (2012). Planning with markov decision processes: An AI perspective. Synthesis Lectures on Artificial Intelligence and Machine Learning, 6(1):1–210.

[13] Bonet, B. and Geffner, H. (2001) 'Planning and control in artificial intelligence: A unifying perspective', Applied Intelligence, 14(3), pp. 237–252. doi: 10.1023/A:1011286518035.

[14] Deisenroth, M. P. and Rasmussen, C. E. (2011) 'PILCO: A model-based and dataefficient approach to policy search', Proceedings of the 28th International Conference on Machine Learning, ICML 2011, pp. 465–472. [15] Keller, T. and Helmert, M. (2013). Trial-based heuristic tree search for finite horizon MDPs.

[16] Bonet, B. and Geffner, H. (2003) 'Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming', Update, (Section 2), pp. 12–21. Available at: http://www.aaai.org/Papers/ICAPS/2003/ICAPS03-002.pdf.

[17] D.P. Bertsekas, Distributed dynamic programming, lEEE Trans. Autom. Control 27 (1982) 6 I O-6 16.

[18] Barto, A. G., Bradtke, S. J. and Singh, S. P. (1995) 'Learning to act using real-time dynamic programming', Artificial Intelligence, 72(1–2), pp. 81–138. doi: 10.1016/0004-3702(94)00011-O.

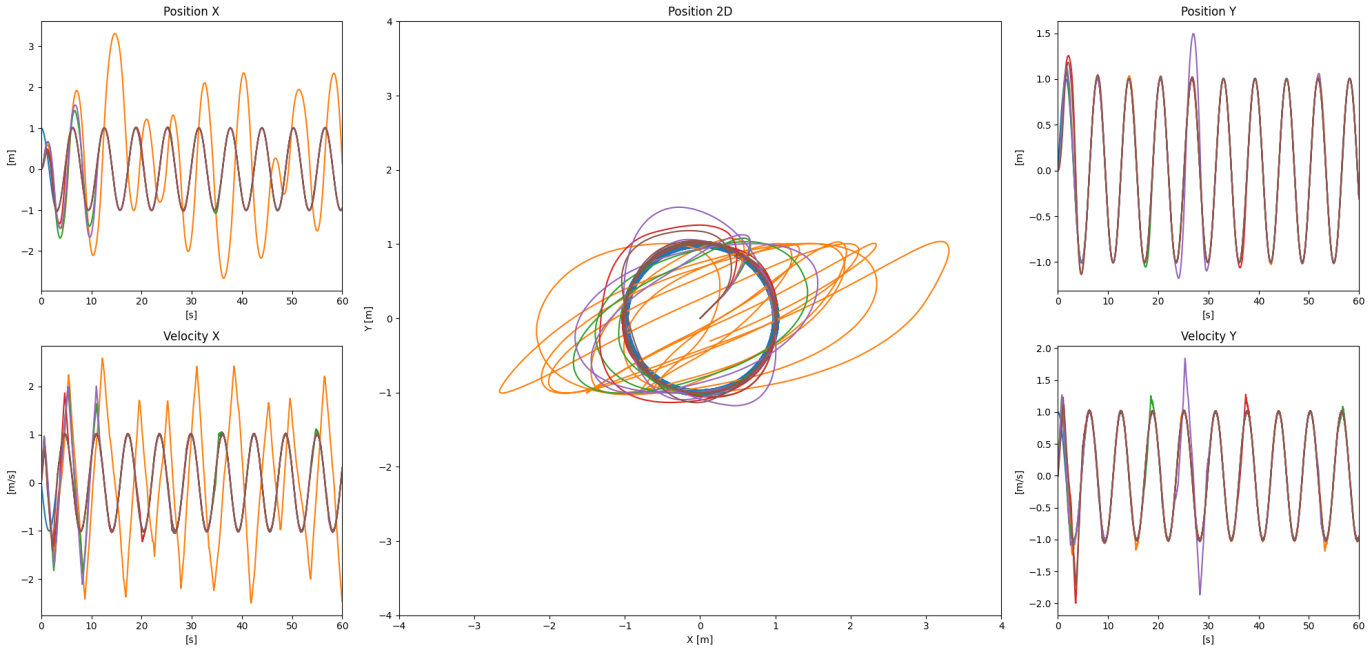

Fig. 9. 2D elliptic Uncoupled, horizon =2, PA = 4, Noise = 0.03, Pos. actions = 3

[19] Mohiuddin, Abdullah; Taha, Tarek; Zweiri, Yahya; Gan, Dongming. 2019. "UAV Payload Transportation via RTDP Based Optimized Velocity Profiles" Energies 12, no. 16: 3049. https://doi.org/10.3390/en12163049

[20] Delamer, J.-A. (2019) 'Planification de strat´egies de navigation et de guidance pour des drones autonomes dans des milieux encombr´es'.

[21] Buffet, O. et al. (2010) 'Planning with Robust (L) RTDP Olivier Buffet , Douglas Aberdeen To cite this version: HAL Id: inria-00509352 Planning with Robust (L) RTDP', (L).

[22] Baier, Hendrik Kaisers, Michael. (2021). 'Novelty and MCTS'.

[23] Feldman, Zohar Domshlak, Carmel. (2013). 'Monte-Carlo Planning: Theoretically Fast Convergence Meets Practical Efficiency'. Uncertainty in Artificial Intelligence - Proceedings of the 29th Conference, UAI 2013.

[24] Baier, Hendrik Winands, Mark. (2012). 'Nested Monte-Carlo Tree Search for online planning in large MDPs'. Frontiers in Artificial Intelligence and Applications. 242. 109- 114. 10.3233/978-1-61499-098-7-109.

[25] Browne, Cameron Powley, Edward Whitehouse, Daniel Lucas, Simon Cowling, Peter Rohlfshagen, Philipp Tavener, Stephen Perez Liebana, Diego Samothrakis, Spyridon Colton, Simon. (2012). 'A Survey of Monte Carlo Tree Search Methods'. IEEE Transactions on Computational Intelligence and AI in Games. 4:1. 1-43. 10.1109/TCIAIG.2012.2186810.

[26] Keller, Thomas Helmert, Malte. (2013). 'Trial-based Heuristic Tree Search for Finite Horizon MDPs'. ICAPS 2013 - Proceedings of the 23rd International Conference on Automated Planning and Scheduling.

[27] Hostetler, J. Fern, A. Dietterich, T.. (2014). 'State aggregation in Monte Carlo tree search'. Proceedings of the National Conference on Artificial Intelligence. 4. 2446-2452.

[28] Li, Lihong Walsh, Thomas Littman, Michael. (2006). 'Towards a Unified Theory of State Abstraction for MDPs'. Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics.

[29] Chen, Guanting Gaebler, Johann Peng, Matt Sun, Chunlin Ye, Yinyu. (2021). 'An Adaptive State Aggregation Algorithm for Markov Decision Processes'.

VII. Appendix

To access the codes and files generated in the simulations, access gitHub at the link here