**Noname manuscript No.**
(will be inserted by the editor)

# A Dynamically Feasible Fast Replanning Strategy with Deep Reinforcement Learning

**Mehmet Hasanzade · Emre Koyuncu**

Received: date / Accepted: date

**Abstract** In this work, we aim to develop a fast trajectory replanning methodology enabling highly agile aerial vehicles to navigate in cluttered environments. By focusing on reducing complexity and accelerating the replanning problem under strict dynamical constraints, we employ the b-spline theory with local support property for defining the high dimensional agile flight trajectories. We utilize the differential flatness model of an aerial vehicle, allowing us to directly map the desired output trajectory into input states to track a high dimensional trajectory. Dynamically feasible replanning problem is addressed through regenerating the local b-splines with control point reallocation. As the geometric form of the trajectory based on the location of the control points and the knot intervals, the control point reallocation for fast replanning with dynamical constraints is turned into a constrained optimization problem and solved through deep reinforcement learning. The proposed methodology enables generating dynamically feasible local trajectory segments, which are continuous to the existing, hence provides fast local replanning for collision avoidance. The DRL agent is trained with different environmental complexities, and through the batch simulations, it is shown that the proposed methodology allows to solve fast trajectory replanning problem under given or hard dynamical constraints and provide real-time applicability for such collision avoidance applications in agile unmanned aerial vehicles. Hardware implementation tests of the algorithm with the agile trajectory tracker to a small UAV can bee seen in the following video link: https://youtu.be/8IiLQFQ3V0E.

M. Hasanzade
Controls and Avionics Research Group, Aerospace Research Center (ITU ARC)
Istanbul Technical University, Istanbul, Turkey
E-mail: abdullahmeh@itu.edu.tr

E. Koyuncu
Department of Aeronautics Engineering
Controls and Avionics Research Group, Aerospace Research Center (ITU ARC)
Istanbul Technical University, Istanbul, Turkey

**Keywords** Agile Unmanned Aerial Vehicles · Trajectory Replanning · Deep Reinforcement Learning

## 1 Introduction

Not long ago, the operations or the applications requiring high-performance guided and navigation would have required the use of tactical-size unmanned aerial vehicles. The main reason for this was that high-performance algorithms required bigger or heavier avionics with high computing capabilities or reliable communication buses linked with the ground systems. However, with the development of technology, these capabilities can now be achieved in smaller avionics, making it possible onboard for small-size unmanned aerial vehicles. New lightweight sensory systems enabled small unmanned systems to have advanced "situational awareness" and allow them to be capable of performing complex missions. Yet, guidance, navigation, and motion planning methodologies are still mostly "conservative" and "use-case-specific," render the UAVs incapable of performing multipurpose-operations.

Many studies have been published on navigating drones in an unknown cluttered and highly dynamic environment. In this vast literature, path-planning algorithms can be grouped under two main headings: optimization-based methods and methods utilizing motion primitives. Optimization-based trajectory planning and re-planning are one of the approaches to generate safe non-collision trajectories, and some of these approaches are using differential flatness to formulate the output trajectory. This approach provides convenience to optimize dynamically feasible trajectories [1–3]. [4] also includes the cost of the maneuvers to find a feasible trajectory. On the other hand, there are other approaches where using sampling-based trajectory generation algorithms to find a pass through waypoints and formulate an optimization problem to find the desired minimum cost trajectory [5]. [6] uses another sampling-based method RRT* and fit these points a polynomial trajectory. This polynomial trajectory is formulated as a quadratic program problem to find a minimum snap trajectory. This polynomial trajectory approach is used as the kinodynamic planning method [7]. To guarantee safety, [8] uses the polyhedral decomposition of the visible free space to convexify obstacle-free space. Identifying the free flight corridors and utilize it as an occupancy grid map is another approach to find a minimum snap trajectory [9]. [10] utilizes point cloud map and uses this in the nearest neighbor search in KD-tree and adopts a sampling-based pathfinding method to generate a flight corridor with safety guaranteed. To generate a path, minimize jerk polynomial trajectory method is used, and the problem is formulated as a quadratically constrained quadratic programming [10]. Some studies use B-splines to define trajectories and optimize control point locations to find feasible trajectories [11]. [12] presents the Bezier curve representation for trajectories. For collision avoidance, the presented algorithm changes the shape of the planned curve by adding an appropriate detour.

Another preferred approach is using motion primitives for online re-planning on small UAV platforms since they can be executed fast, and each primitive is generated to be dynamically feasible. [13] uses primitives to evaluate maneuvers probabilistically for collision avoidance, chooses a maneuver based on unconstrained objective combining collision avoidance and navigation. [14] also uses a probabilistic approach as robust motion primitives to overcome uncertainties. [15] samples the vehicle's control space to generate motion primitives to guarantee the trajectory to be dynamically feasible. Another approach is to generate minimum jerk primitives online given in vehicles' current state and desired final state [16]. [17, 18] uses a maximum dispersion algorithm to use pre-computed motion primitives. [19] generates motion primitives off-line automatically as a robot motion model and uses an incremental re-planning algorithm that allows producing smooth, dynamically feasible motion plans while reusing previous computation.

In addition to these approaches, [20] presents an efficient perception and planning approach, which uses the Triple Integrator Planner (TIP). This approach allows the system to operate at its physical limits. Also, [21] presented an improved method of TIP called Relaxed-constraint Triple Integrator Planner (R-TIP) that overcomes the perception system used onboard that cause limiting vehicle speed. The proposed method solves this problem by choosing a previously generated motion primitive.

In this study, we propose a fast re-planning strategy based on deep reinforcement learning for highly agile aerial vehicles. First, we utilize the differential flatness model of an air vehicle, allowing us to directly map the desired output trajectory, which is parameterized with b-spline curves, into required input states to track trajectory. Moreover, we use a perception model with fixed range and FOV on the vehicle, and as soon as the vehicle detects the obstacle, it performs the real-time evasive action through repetitive re-planning over an infinite trajectory. Specifically, the algorithm is initialized with a flight trajectory plan, then performs optimal control point vector update and knot insertion to generate a dynamically feasible conflict-free trajectory. Through this modification, the regenerated trajectory provides feasible evasive maneuvers for the vehicle, where the location and the number of the added control points form the "agility" of this evasive maneuver. The control point insertion considering dynamic constraints and the defined agility metrics is transformed into a trajectory optimization problem, which is solved through deep reinforcement learning (DRL). We utilized the proximal policy optimization (PPO) method to train the re-planner with the random forest generation environment. The agent produces re-planned dynamically feasible conflict-free trajectories with modified control points approximately in 400us, which enables the real-time flight trajectory generation for highly agile aerial vehicles.

The rest of the paper organized as follows: Section-2 explains the B-spline and properties used in output trajectory formulation. In Section-3, we explain the differential flatness model for the vehicle. Section-4 introduces deep reinforcement learning and proximal policy optimization for the control point vector update. In Section-5, we present several experimental results from the

rigorous simulations. Finally, we summarise the methodology and give a conclusion and future works at the end of the paper.

## 2 Differential Flatness based Dynamic Model

Considering the aerial vehicles, the trajectory planning problem with dynamical constraints on the vehicle might become extremely challenging due to their high dimensional dynamics. However, for most of the dynamical systems, it is generally possible to parameterize a part of the state regarding a given output trajectory and its time derivatives. This phenomenon called differentially flatness enabling an effective dimension reduction when the whole state and the input can be parameterized with one output. Let the state of the system is $x \in \mathbb{R}^n$ and let the input of the system is $u \in \mathbb{R}^m$. A nonlinear system $\dot{x} = f(x, u)$, $y = h(x)$ is differentially flat, if one can write the system equation in the following form:

$$z = \zeta(x, u, \dot{u}, ..., u^{(p)}) \tag{1}$$

where,

$$x = x(z, \dot{z}, ..., z^{(q)}) \tag{2}$$

$$u = u(z, \dot{z}, ..., z^{(q)}). \tag{3}$$

Through differential flatness formulation, all of the feasible trajectories for the system can be written as functions of a flat output $z \in \mathbb{R}^m$ and its derivatives [2].

### 2.1 Aerial Vehicle Model

In this work, the differential flatness principle is applied to real-time flight trajectory generation problem of an aerial vehicle. We have formulated the desired output trajectory through b-spline curves, enabling a map into the required input states to track the given trajectory. For the implementation purposes, we utilized dynamical model of a quadcopter, which can be derived by the Lagrangian approach as given below:

$$
\begin{aligned}
\ddot{x} &= U_1 \frac{(\cos\psi)\cos\phi\sin\theta + \sin\psi\sin\phi}{m} \\
\ddot{y} &= U_1 \frac{(\sin\psi)\cos\phi\sin\theta - \sin\phi\cos\psi}{m} \\
\ddot{z} &= U_1 \frac{(\cos\theta)\cos\phi}{m} - g \\
\dot{p} &= \frac{I_y - I_z}{I_x} qr + \frac{1}{I_x} U_2 - \frac{J_m}{I_x} q\Omega_r \\
\dot{q} &= \frac{I_z - I_x}{I_y} pr + \frac{1}{I_y} U_3 + \frac{J_m}{I_y} p\Omega_r \\
\dot{r} &= \frac{I_x - I_y}{I_z} pq + \frac{d}{I_z} U_4 \\
\dot{\phi} &= p + \sin\phi\tan\theta q + \cos\phi\tan\theta r \\
\dot{\theta} &= \cos\phi q - \sin\phi r \\
\dot{\psi} &= \frac{\sin\phi}{\cos\theta} q + \frac{\cos\phi}{\cos\theta} r
\end{aligned}
\tag{4}
$$

Where $m$ is the aircraft mass, $p$, $q$ and $r$ are the angular rates in the body-frame, $I_x$, $I_y$ and $I_z$ are the moments of inertia. $J_m$ is the motor inertia, and $\Omega_R$ is defined as $\Omega_R = -\Omega_1 - \Omega_3 + \Omega_2 + \Omega_4$. The input vector with $U_1, U_2, U_3, U_4$ is expressed in terms of the motor angular rates $\Omega_i$, where $i = \{1, 2, 3, 4\}$ and given as follows:

$$
\begin{aligned}
U_1 &= b\sum_{i=1}^{4} \Omega_i^2, \\
U_2 &= bl(\Omega_4^2 - \Omega_2^2), \\
U_3 &= bl(\Omega_3^2 - \Omega_1^2), \\
U_4 &= d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2)
\end{aligned}
\tag{5}
$$

An aerial vehicle state can be represented by the flat variables $\boldsymbol{x} = [x, y, z, \psi]$, where $x, y, z$ are the Cartesian positions and $\psi$ is the vehicle's yaw angle. With these four inputs, the dynamics of the quadcopter can be expressed as differentially flat. To obtain these flat outputs, we utilize the b-spline curves to formulate the Cartesian position vector $[x, y, z]$ and derivatives. Yaw will be considered as a constant for simplification. Therefore, the state of the aerial vehicle $\boldsymbol{x}$ is given as follows:

$$
\boldsymbol{x} = [x^T \dot{x}^T \ddot{x}^T]^T = [x^T v^T a^T]^T \tag{6}
$$

Through B-spline representation, which will be explained in Section-3, one can define position of the vehicle and its derivatives by using the B-spline continuously differentiable property. The total thrust $U_1$ can be represented by the flat outputs as follows:

$$U_1 = m\sqrt{\ddot{x}^2 + \ddot{y}^2 + (\ddot{z} + g)^2} \tag{7}$$

The pitch $\theta$ and roll $\phi$ angle equations is given as follows:

$$\phi = \arcsin m\ddot{x}\sin\psi - m\ddot{y}\cos\psi/U_1 \tag{8}$$

$$\theta = \arcsin \frac{m\ddot{x}\cos\psi - m\ddot{y}\sin\psi}{U_1\cos\phi} \tag{9}$$

Finally, $U_2$, $U_3$ and $U_4$ can be derived as in Eq. (7) defined through flat outputs and Lagrangian model of quadcopter.

### 2.2 Perception Model

Typically, aerial vehicles have limited range and field of view (FOV). Because of the forward and horizontal acceleration that the air vehicle can produce, the sensor needs to have more FOV, on the other hand, in that physically maximum acceleration situations, there is no way that air vehicle can sense the obstacles around itself [21]. The generating infinite collision-free trajectory for a flight in clutter environments with a limited field of view, regardless of the planning algorithm, resembles the flying in an environment with a random obstacle generating process. In [22], it is shown that when this process is ergodic, the existence of an infinite collision-free trajectory exhibits a phase transition with certain critical speed. Sensing with random obstacle process is depicted in Fig.1.

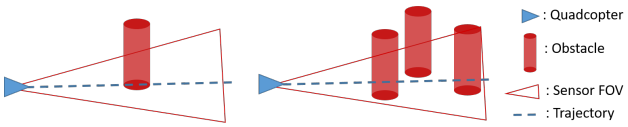



Fig. 1: Sensing with limited field of view (FOV) and range for the aerial vehicle. The local interest of environment is perceived as random.

## 3 Trajectory Characterization and Modification

In this section, we provide the details about output trajectory parameterization for the aerial vehicle, and how to provide modification over the generated trajectory.

### 3.1 Trajectory Parameterization with B-Spline

The B-Spline representation allows one to describe any flight trajectory with their derivatives, which enables parameterizing of a flat output through the generation of several joined polynomials. Generally, a *pth*-degree B-Spline curve is defined as follows:

$$p(t) = \sum_{i=0}^{n} P_i B_{i,p}(t) \qquad a \leq t \leq b \tag{10}$$

Where $p(t)$ denotes the curve at $t$ and the $P_i$ are the control points. The $B_{i,k}(t)$ are basis functions that can be computed using the De Boor-Cox recursive formula [23–25].

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$B_{i,p}(t) = \frac{u - u_i}{u_{i+p} - u_i} B_{i,p-1}(t) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} B_{i+1,p-1}(t) \tag{12}$$

These basis functions are defined as the function of the knot vectors:

$$\tau = [u_0, ..., u_m] \tag{13}$$

We used uniform knot vector which could also be presented as:

$$\tau = [a, ..., a, t_{p+1}, ..., t_{m-p-1}, b, ..., b] \tag{14}$$

The length of the knot vector is $m + 1$ where $m = n + p + 1$ [25]. We assume that $a = 0$ and $b = 1$ for the unity knot vector, and the first and last knots have multiplicity $p + 1$. We define a knot vector $\tau = [t_0, ..., t_m]$ is uniform if all interior knots are equally spaced such that $d = t_{i+1} - t_i$ for all $p \leq i \leq m - p - 1$. .

**Property 1.** ***Endpoint interpolation***
The trajectory $p(t)$ with control point array $[P_0, ..., P_n]$ consisting of $n + 1$ control points, and by assuming first and last knots have multiplicity $p + 1$ where $a = 0$ and $b = 1$, then it holds *endpoint interpolation* property such that $P_0 = p(0)$ and $P_n = p(1)$.

The $p(t)$ curve has a strong relationship between the instantaneous positions of the vehicle as the $p(t)$ curve is the normalized form of the generated trajectory. The *endpoint interpolation* ensures that the first and last control points are the initial and final states of the aerial vehicle.

Let $p_i^{(k)}$ denote the $k$th-derivative of $p_i$. Then $p(t)$ said to be $C^j$ continuous at the break-point $t_i$ if $p_i^{(k)}(t_i) = p_{i+1}^{(k)}(t_i)$ for all $0 \leq k \leq j$.

**Property 2.** ***Continuity***
The trajectory $p(t)$ is infinitely differentiable in the interior of knot intervals, and it is at least $p-k$ times continuously differentiable at a knot multiplicity $k$. Then, typically a $pth$-degree B-Spline curve includes piecewise polynomials of degree $p$ and have $C^{p-1}$ continuity. The derivatives of $p(t)$ curve enables to define the velocity, acceleration, jerk and snap of the trajectory, and can be expressed as follows:

$$p^{(k)}(t) = \sum_{i=0}^{n} P_i B_{i,p}^{(k)}(t) \qquad t \in [0,1] \tag{15}$$

where $k$ represents the order of derivatives. B-spline curve is a linear combination of the basis function $B_{i,p}(t)$,therefore the differentiability and continuity of the B-spline depends on their basis functions $B_{i,p}(t)$. The Eq. (15), allow us to obtain velocity and acceleration vectors of the trajectory through first and second derivatives respectively, and an example trajectory with velocity and acceleration vectors is given in Fig. 2.

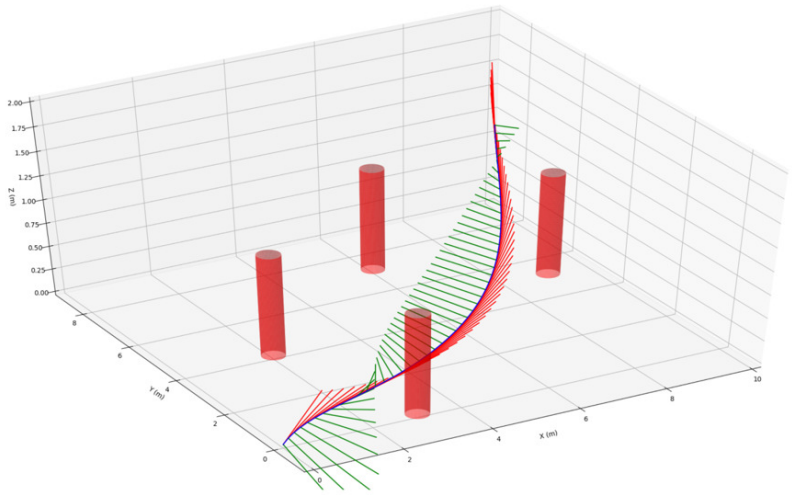



Fig. 2: An example B-spline trajectory agility with instantaneous velocities(red) and accelerations (green), where the velocity vectors shows the motion direction of the aerial vehicle.

Considering the agility in generated flight trajectories, in addition to velocity and acceleration continuity, we have included jerk and snap continuity

as well while defining flat outputs to track; therefore, we have chosen to represent flat output trajectories with $p = 6$-degree B-splines to ensure at least $C^4$ continuity.

The B-spline curves have a strong *convex hull property* that the curve is constrained in the convex hull of its control polygon.

**Property 3.** ***Strong convex hull***
The curve is contained in the convex hull of its control polygon. If $t \in [t_i, t_{i+1}]$ where $p \leq i \leq m - p - 1$, then $p(t)$ is in the convex hull of the control points $P_{i-p}, ..., P_i$. In other words, the generated trajectory remains within certain limits, which are formed by control points $P_i$ and as a result of this known beforehand. Knot insertion or breaking uniformity, therefore, does not change these limits of this convex form. This property is depicted in Fig. 3 with an example.

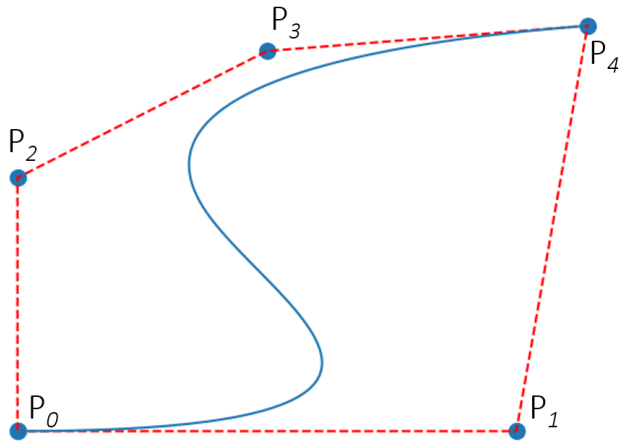



Fig. 3: An example of B-spline curve defined by five control points $P_{0,..,4}$ which are shown as blue dots. The curve is completely enclosed within the convex hull created by its control points.

The other property that we exploit is *local support*.

**Property 4.** ***Local support***
Relocating $P_i$ changes $p(t)$ trajectory only interval of $[t_i, t_{i+p+1})$ due to the fact that $B_{i,p} = 0$ for $t \notin [t_i, t_{i+p+1})$. In other words, relocating one of control points changes the curve's position and derivatives only locally [25]. This property allows us to relocate the control points, without repetitively checking the collision and dynamical feasibility of the whole trajectory with its positions and derivatives.

One can note that there is a strong relationship between the knots $t$ and the time allocations over the generated trajectory as $t_i \in [0, 1] \quad \forall i \in \mathbb{R}$ is the normalized time parameter. In the other words, considering the knot vector

$\eta\tau = \eta[t_1, ..., t_k]$, the $\eta$ factor does not change the geometry of generated trajectory while scaling the derivatives of it.

### 3.2 Trajectory Modification with Control Point Relocation and Knot insertion

The trajectory replanning strategy is based on control point relocation and knot insertion to the generated B-spline curve. Through exploiting the following properties of the B-splines, we can achieve this replanning with knowing the limits and the local interest of the modification over the trajectory.

After defining the trajectory with a uniform B-spline, through control point and knot insertion, one can replan the trajectory only changing the geometry of local interest for certain intents such as dynamic collision avoidance, re-optimization, etc. Let us assume that the collision over the trajectory around $\bar{t}$ sensed, meaning that immediate replanning is required, which is depicted in Fig. 4.

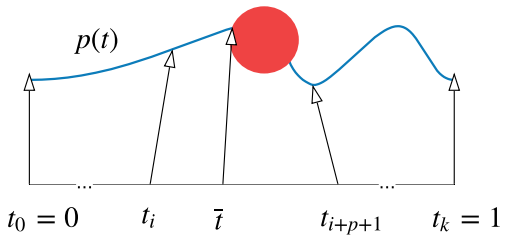



Fig. 4: Collision sensing over trajectory around $\bar{t}$ knot point

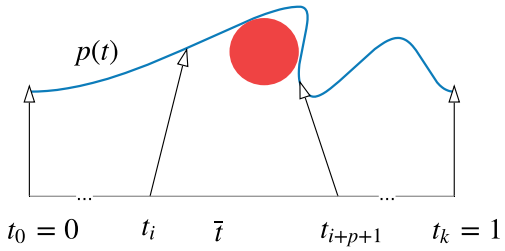



Fig. 5: Control point relocation over the trajectory

*Control Point Relocation*

To express the relocation of the control point, Eq. (10) can be written in the following form :

$$p(t) = P_0 B_{0,p}(t) + ... + P_i B_{i,p}(t) + ... + P_n B_{n,p}(t) \tag{16}$$

Where $0 < i \leq n$. Let us assume to relocate the $P_i$, then the new control point and location become $\bar{P}_i$:

$$\bar{P}_i = P_i + V \tag{17}$$

Where $V$ presents the relocation vector. After relocation, the B-spline curve equation becomes:

$$\bar{p}(t) = P_0 B_{0,p}(t) + ... + (P_i + V) B_{i,p}(t) + ... + P_n B_{n,p}(t) \tag{18}$$

or

$$\bar{p}(t) = p(t) + V B_{i,p}(t) \tag{19}$$

The relocation process, due to the *local support*, only effects the curve within the $t \in [t_i, t_{i+p+1})$ interval where $B_{i,p}(t) \neq 0$ and this is depicted in Fig. 5.

*Knot Insertion*

Relocating existing control points gives us a limited behavior for replanning and in some cases, might cause additional agility, almost breaking dynamic feasibility. By addressing this problem, the knot insertion methodology shown in Fig. 6, enables an increasing number of control points; in other words, more flexibility, without changing the geometry of the trajectory.

Let us assume that $p(\bar{t})$ is the closest point to the sensed obstacle or mean of the sensed obstacles $p_{so}$ on the trajectory. $\bar{t}$ is starting from the current time $t_0$ when the aerial vehicle sense any object with its sensor. Then, it uses the knot insertion method to add this new $P_{new}$ control point.

$$P_{new} = p(\bar{t}) \tag{20}$$

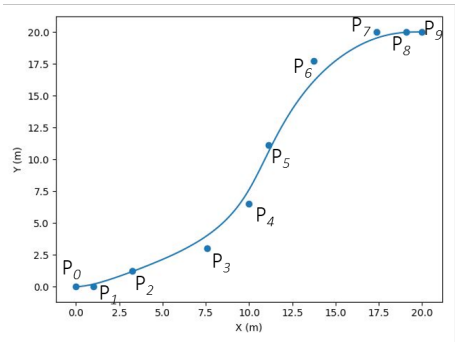
$$\bar{t} = min(\|p(\bar{t}) - p_{so}\|_2) \tag{21}$$

$$\bar{t} \in [t_0, 1]$$

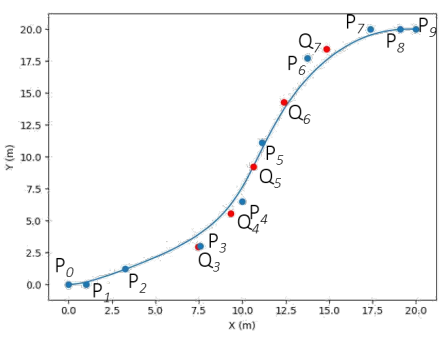
As we already know the location of $P_{new}$ from (20), we need to insert $\bar{t}$ inside the knot interval from the knot vector. We assign $l = t_i$, where $\bar{t} \in [t_i, t_{i+1}]$. Then, we utilize following Eq. (22) and Eq. (23):

$$\alpha_i = \begin{cases} 1 & i \leq l - p + 1 \\ 0 & i \geq l + 1 \\ \frac{\bar{t} - t_i}{t_{l+p+1} - t_i} & l - p + 2 \leq i \leq l \end{cases} \tag{22}$$

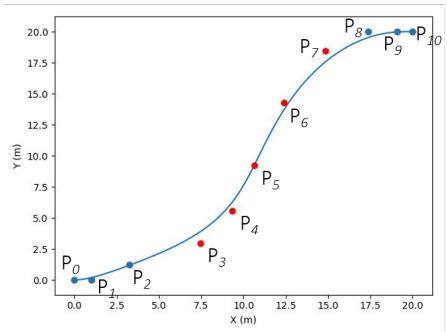
$$\hat{P}_i = (1 - \alpha_i)P_{i-1} + \alpha_i P_i \tag{23}$$




(a) Initial Trajectory




(b) Knot insertion method




(c) New control points replaced

Fig. 6: Replacing control points with the new ones through knot insertion method is presented. (a) is a simple trajectory defined by $P_0, ..., P_9$ control points. (b) shows knot insertion method and the output of the method is shown as $Q_3, ..., Q_7$ control points. (c) presents the result of the knot insertion method where $P_3, ..., P_6$ are replaced with $Q_3, ..., Q_7$ control points.

Through the knot insertion, we find the knots correspond to the required control point positions. We replace control points $P_{l-k+1}, ..., P_{l-1}$ with $Q_{l-k+1}, ..., Q_l$; hence, the new control point vector is defined as $\hat{P}_i$, and the new B-spline curve can be found with following Eq. (24).

$$p(t) = \sum_{i=0}^{n+1} \hat{P}_i \hat{B}_{i,p}(t) \qquad t = [0, 1] \tag{24}$$

An example control point vector modification through knot insertion is depicted in Fig. 6. It should be noted that an increasing number of control points does not cause geometric change over the trajectory.

### 3.3 Replan Decision Point

In trajectory planning problems, mostly global and local planning/replanning runs together to achieve safe navigation in a highly dynamic environment. In addition to geometric and dynamic feasibility, local planners require computational feasibility that considers computational and time complexity of the utilized algorithms. To achieve real-time fast replanning, the algorithm should consider calculation time as the vehicle flies over the trajectory. Through the *local support* property as we know already, which segment of the trajectory to be reformed, it is essential to consider the backpropagation of this modification over the planned trajectory.

Now suppose that $\bar{t}$ is the knot vector value with $\bar{t} \in [t_k, t_{k+p+1})$ and $p(\bar{t}) = P_i$ is the control point where a modification requires and the obstacle detection occurred at $p(t_0)$. Recall that knot values over the trajectory has a direct scaling relationship between the time as the knot vector $t \in [0, 1]$ is the normalized time value. It is possible to define an algorithm run-time and transform into a knot value, let it be $\delta_t$. Due to the translation of $P_i$ to $\bar{P}_i$, all trajectory points outside the $\bar{t} \in [t_k, t_{k+p+1})$ interval are to be unaffected. Therefore following safety rule should be satisfied always while replanning;

$$t_k \geq t_0 + \delta_t, \quad \bar{t} \in [t_k, t_{k+p+1}) \tag{25}$$

This rule guarantees that any possible replanning trajectory has a chance to provide an evasive maneuver. As the replanning $[t_0, t_{p+1})$ span over the trajectory depends on the size of the knot vector, the scope of the replanning segment shows a difference with the number of control points. There is a trade-off here between the trajectory description resolution and the number of fails in replanning algorithm run: the more control points providing more flexibility in trajectory modification means possibly more fails in dynamic feasibility check; while fewer control points give smoother trajectory modifications, unlikely breaking the dynamic feasibility over the trajectory. Fig.7 demonstrates these effects with different trajectory description resolutions.

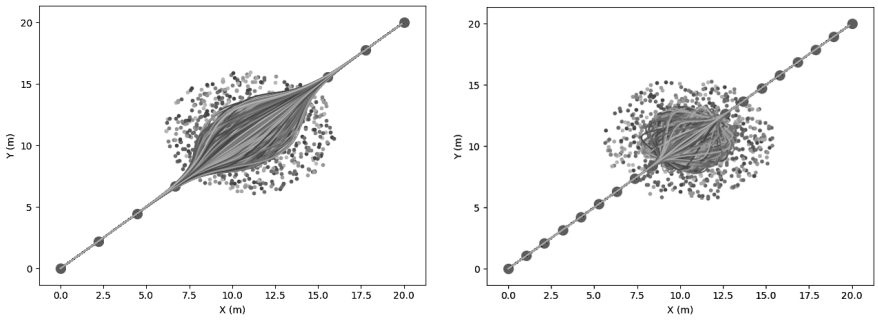



Fig. 7: Replanning scopes over the trajectories with different number of control point representations

## 4 Optimal Replanning with Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a tool that can provide an optimal decision based on what it has learned in certain situations. The purpose of this tool, which uses Markov decision processes (MDP), is to produce an action based on the state of the agent in a particular environment. After taking any action, the agent receives a reward for the action and the next state. Through estimating/learning the rewards corresponding to the state, the agent develops its policy.

In reinforcement learning, the maximization of the accumulated reward is the main goal of the agent. Therefore, In each episode, the agent learns to self-adjust policies to maximize the accumulated reward. The agent learns these policies through interaction with the unknown environment, which is formulated as an MDP by a tuple $M = (S, A, T, R, \gamma)$. $S$ is a set of agent states, and $A$ is a set of actions. This representation assumes that the next state $s_{t+1}$ is only conditional on the current state $s_t$ and action $a_t$ which is derived from Markov property. $T$ is a transition probability function $T : SxA \in [0, 1]$, which maps the transition to probability.$R : SxA \in R$ is the reward function represents the amount of reward or punishment that the environment will pass in for a state transition. $\gamma$ is the discount factor $\gamma \in (0, 1]$, which is used to receive the return from the process as a sum of the discount rewards. Further thought, the agent is at state $s_t$, and it takes action $a_t$ and receives a reward $r_t$ based on predefined reward function. Then, the environment transitions to state $s_{t+1}$ according to the $T$.

Reinforcement learning applies this MDP formulation and modifies it to use for learning. The technique that we use is defined under policy-based reinforcement learning. This policy represents a function mapping a state to action, and the agent aims to optimize the policy to maximize the accumulated reward.

### 4.1 Proximal Policy Optimization

The Proximal Policy Optimization (PPO) is one of the policy gradient methods for reinforcement learning. It uses the first-order algorithm to utilize the benefits of trust region policy optimization (TRPO) such as reliable performance and data efficiency [26]. The TRPO aims to maximize the surrogate objective function $L^{CPI}$ (conservative policy iteration) [27]. Let $r_t(\theta)$ is the probability ratio shown as follows:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \qquad r(\theta_{old}) = 1 \tag{26}$$

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\big[r_t(\theta)\hat{A}_t\big] \tag{27}$$

$CPI$ refers to conservative policy iteration. [26] proposes new objective function, which is presented in below:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\big[\min(r_t(\theta)\hat{A}_t, clip(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\big] \tag{28}$$

where epsilon $\epsilon$ is a hyperparameter that we have chosen as 0.2. The difference between TRPO and PPO based on the $clip(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ term, which allows us to adjust the surrogate objective by saturating the probability ratio. The reason for this clip function is to avoid extensive updates by the maximizing $L^{CLIP}$. Clipping function also depends on the interval $[1-\epsilon, 1+\epsilon]$ term, where we can change the size of the update rate by changing the $\epsilon$.

For the training part of the DRL, first, we need to define observation, reward, and action vectors. In order to start the learning process, random scenarios where the agent to be trained are generated. In Fig.8, some possible scenarios are depicted. In each scenario, Poisson distributed random numbers are generated to create the number of obstacles encountered by air vehicles through the distribution equation given by Eq. (29), and the position of the obstacles is randomly assigned according to a uniform distribution.

$$Pois(x;\mu) = (e^{-\mu})(\mu^x)/x! \tag{29}$$

Considering the small region of interest due to limited FOV, we only produce one collision situation in each scenario during the training. Instead of running the training algorithm depending on the cumulative rewards where there are too many obstacles, we have created specific scenarios that only locates one or more obstacles once, and tries to generate solution locally. This modification simplifies the collision avoidance in the cluttered environment and allows the DRL agent to learn faster.

In DRL, we defined observation, reward and action as follows:

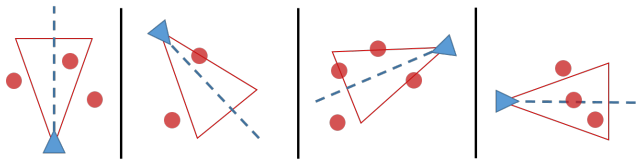


Fig. 8: Depiction of scenarios with random obstacle generation and limited FOV.

*Observation Space:*
We assume that the air vehicle can obtain the state measurements via inertial measurement unit (IMU) sensor and GPS. The trajectory plan, defined through B-spline curve, involves initial and future states and their derivatives. For describing a collision state, the observation state also includes the states of the obstacles that are sensed, and the new inserted control point location. Observation space tuple is given as follows:

$$\Omega = \{p(t), \dot{p(t)}, \ddot{p(t)}, p_{obs}, P_{new}\} \tag{30}$$

where $\{p(t), \dot{p(t)}, \ddot{p(t)}\}$ are the current state and its derivatives of the aerial vehicle over the generated trajectory; $p_{obs}$ is the closest point to collision with the obstacles; and $P_{new}$ is the newly inserted control point.

*Action Space:*
Action space consists of the relocation position with respect to the newly inserted control point. After the DRL agent generates a new action based on the observation space, this action values are added to this new control point, and the B-spline curve is updated. Action space tuple is given with relocation of newly inserted control point $\mathcal{A} = \{P_{new}\}$.

*Reward Function:*
In this local re-planning problem, DRL agent's learning is directly depends on how to define the reward function.

Considering agile flight enabling fast collision avoidance, we have chosen to use the agility metric for the reward function. Given the definition, agility can be a positive reward or penalty to obtain aggressive or smooth flight under the dynamical constraints enabling a feasible trajectory. These constraints limit the search space for the training of the agent for re-planning.

The definition of the agility metric is a well-studied topic on many different vehicles. In [28, 29] many of the agility metrics are summarised. [29] especially focuses on maneuverability and agility for rotary unmanned aerial vehicles, and defines the attitude quickness as $Q = \dot{\alpha_{peak}}/\Delta\alpha$ where $\dot{\alpha_{peak}} = \{p_{peak}, q_{peak}, r_{peak}\}$ and $\Delta\alpha = \{\theta, \phi, \psi\}$. Considering these definitions, we have chosen the agility metric based-on *Instantaneous-rates*, which are also proposed in [30–32]. Hence, the reward function $R$ can be defined with the agility metric as a second derivative of the generalized state variables, as given below:

$$R = \mp \sum_{t=t_i}^{t_{i+p+1}} \sum_{j=1}^{m} \omega_j \ddot{\mathbf{x}}_{\mathbf{j_t}} \tag{31}$$

where $\mathbf{x} \in \mathbb{R}^m$ represents the generalized state variables and $\omega_j$ is the weight value for each state. By setting these weight values, we have chosen to use the agility definition with high dominance at angular velocities.

*Dynamic Feasibility:*

To make sure the generated trajectory is feasible, we must ensure that velocity, acceleration, jerk, and Euler angles at each state of the trajectory remain bounded, and guarantee that the newly generated trajectory remains within the collision-free space with small safety volume, as depicted in Fig.9. Note that relocating one of control point $P_i$ changes $p(t)$ trajectory only interval of $[t_i, t_{i+p+1})$, and requires collision check over this segment of the trajectory only. To meet the dynamical feasibility, we included constraints inside the reward function. If at any point, the generated trajectory respect to the action breaks the following constraints, then the reward function takes a big negative constant value.

$$\|p(t) - p_{so}\|_2 \leq d_{obs,r} + d_{safe} \tag{32}$$

$$v_{min} \leq \dot{p}(t) \leq v_{max} \qquad t \in [t_i, t_{i+p+1}] \tag{33}$$

$$a_{min} \leq \ddot{p}(t) \leq a_{max} \qquad t \in [t_i, t_{i+p+1}] \tag{34}$$

$$j_{min} \leq \dddot{p}(t) \leq j_{max} \qquad t \in [t_i, t_{i+p+1}] \tag{35}$$

$$\theta_{min} \leq \theta(t) \leq \theta_{max} \qquad t \in [t_i, t_{i+p+1}] \tag{36}$$

$$\phi_{min} \leq \phi(t) \leq \phi_{max} \qquad t \in [t_i, t_{i+p+1}] \tag{37}$$

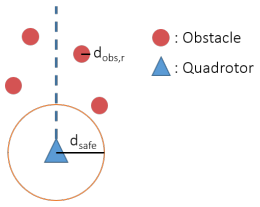



Fig. 9: Safety volume around the vehicle

## 5 Fast Replanning Hardware and Software Implementation Results

To generate a DRL agent, we used OpenAI gym environment [33] with proximal policy optimization. After defining the required spaces, we trained the DRL agent approximately 1200 episodes. Because of the number of CPU and scenario number for each episodes, the DRL agent trained with approximately 30 million scenarios. The reward results can be seen in Fig.10.

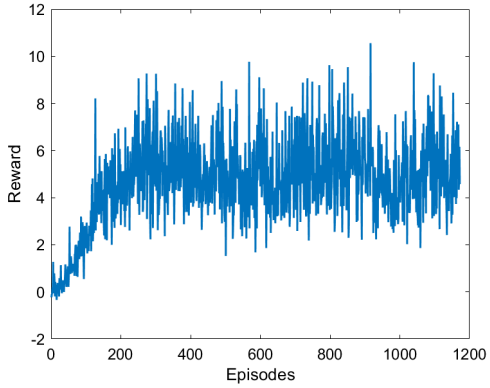



Fig. 10: Reward performance, each episode includes 5120 randomly generated scenarios

The initial point $G_{init}$ and the goal point $G_{goal}$ are randomly generated with a specific distance between them shown in Fig.11. From very start, because there are not any known-obstacles, the minimum distance trajectory between $G_{init}$ and $G_{goal}$ is a simple straight line. Because of randomly generation of these points, the lines heading also changes. This is very important for the scenario generation allows the generalization of the DRL agent. Another important point to mention is encountering obstacles situations.

The on-board sensor model [21] is used to detect obstacles and find their relative positions respect to the air vehicle's location. We used a sensor model on the air vehicle with a 20° field of view (FOV) and 4 meters range. Whenever sensor finds any obstacle, the local re-planning algorithm starts, which consists of adding new control point with knot insertion algorithm, generating observation for DRL agent, action generated by DRL agent and with new location for the new control point is used to update the trajectory. The schematics for the scenario is shown in Fig.11.

In the cluttered environment simulations, the scenarios are designed through randomly cluttered forests with different obstacle density where the vehicle must fly from $G_{init}$ to $G_{goal}$ in $20m \times 20m$ environment. Before the flight, the air vehicle generates a B-spline, which is an almost straight flight trajectory between $G_{init}$ and $G_{goal}$, which can be seen in Fig.11.

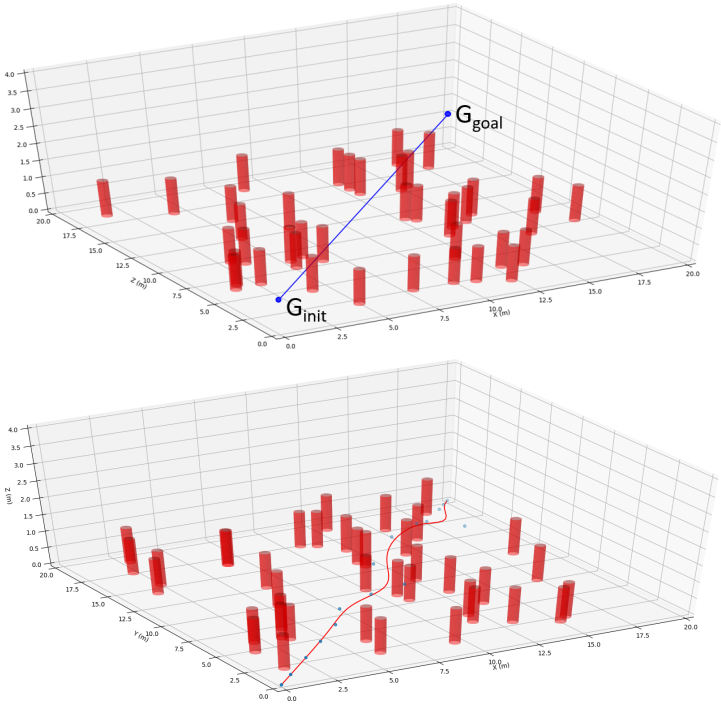
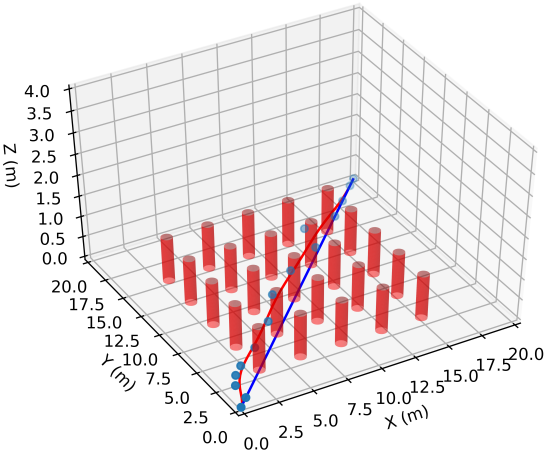



Fig. 11: Replanning algorithm in $20m \times 20m$ environment with an obstacle density of 0.1 $obstacles/m^2$.

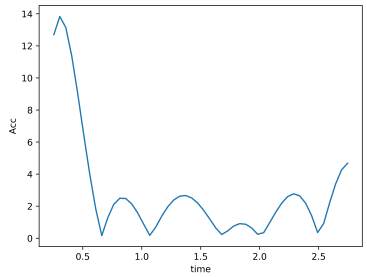
We have conducted several scenarios at different complexities for testing our DRL agent. The results from three randomly generated scenarios can be seen in Fig.12, 13 and 14 such that the obstacle densities are higher than 0.05 $obstacles/m^2$ and 0.1 $obstacles/m^2$.

### 5.1 Batch Simulation Results

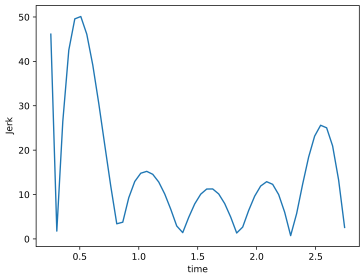
Considering the dynamic feasibility of replanning, through the proposed methodology, we have full control authority over the defined dynamical constraints. The methodology allows us to define the constraints and generate the trajectories to comply with them. To test the ability of the algorithm, e.g., in the training phase of DRL agent, we have defined upper bound constraint at $11m/s$ and lower bound constraint at $8.5m/s$ in velocity; upper bound at $15m/s^2$ in acceleration, and upper bound $50m/s^3$ in jerk. This set of constraints has been chosen for a small UAV to track a trajectory at high speeds based on the strict dynamic limits. Through a batch run with 500 randomly generated scenarios, the velocity, acceleration, and jerk profiles generated by the algorithm, are shown in Fig. 15. As seen in the figure, the algorithm does not break the constraints, even though hits the limits in some cases. In addi-



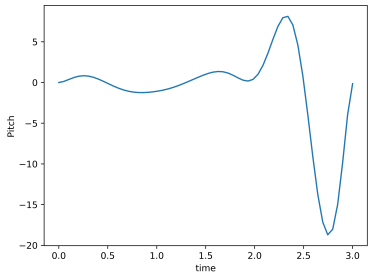

(a) Scenario



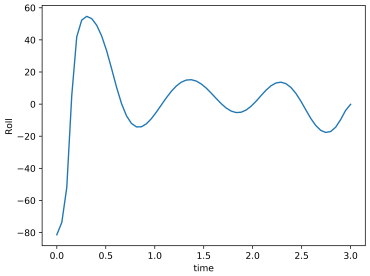

(b) Acceleration




(c) Jerk

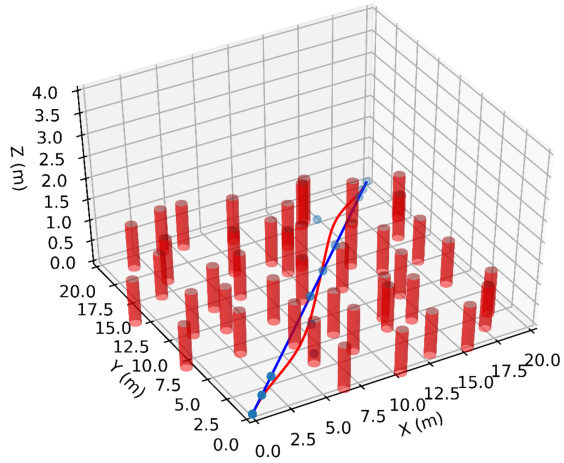



(d) Pitch

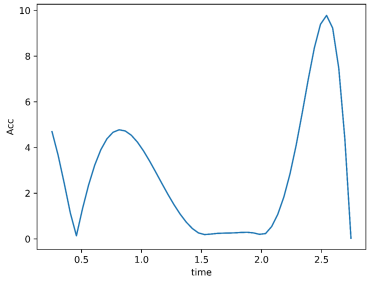



(e) Roll

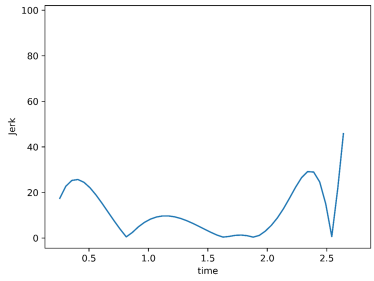
Fig. 12: The cluttered environment simulation, randomly generated obstacles in $20m \times 20m$ environment with an obstacle density of $> 0.05 obstacles/m^2$.



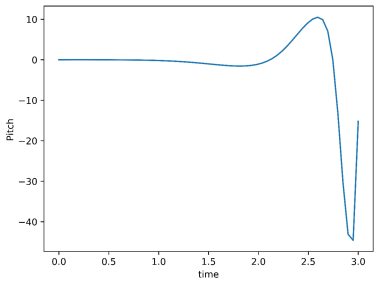

(a) Scenario



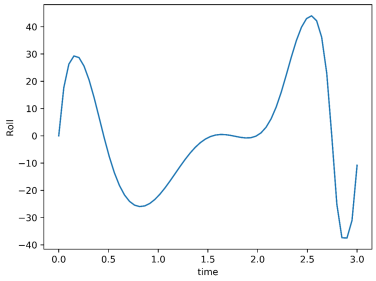

(b) Acceleration




(c) Jerk

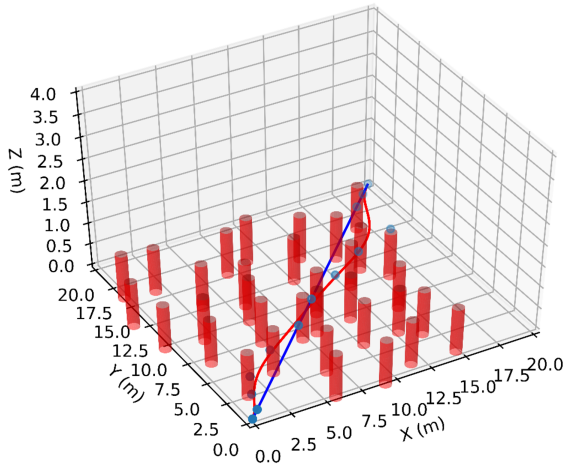



(d) Pitch

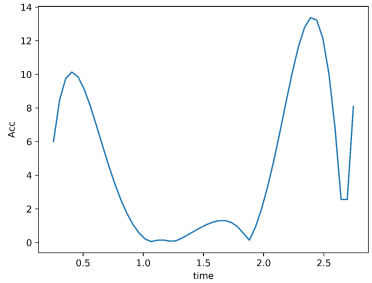



(e) Roll

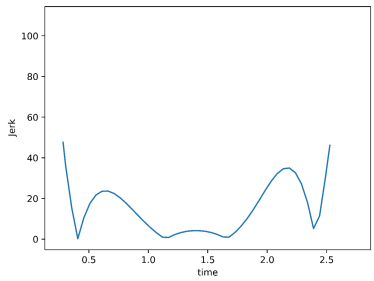
Fig. 13: The cluttered environment simulation, randomly generated obstacles in $20m \times 20m$ environment with an obstacle density of $> 0.1 obstacles/m^2$.



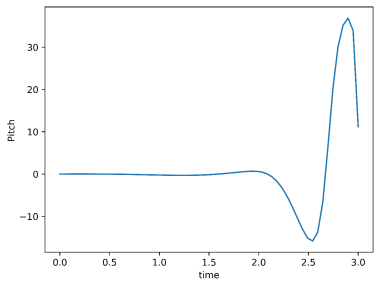

(a) Scenario



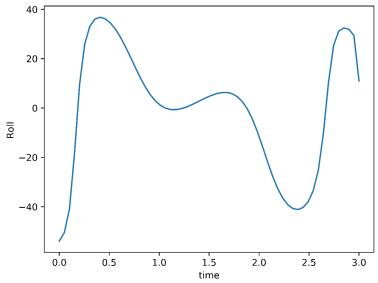

(b) Acceleration




(c) Jerk



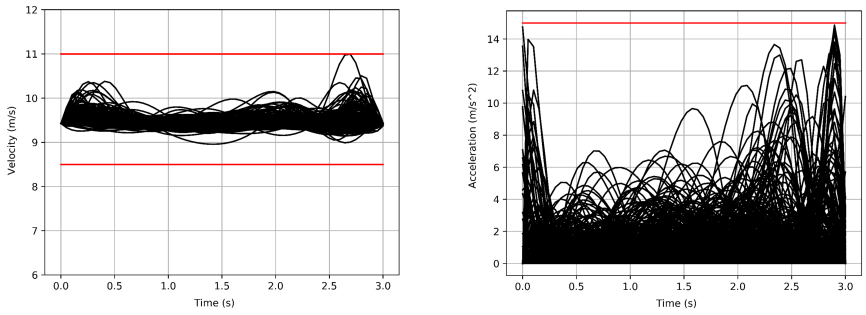

(d) Pitch



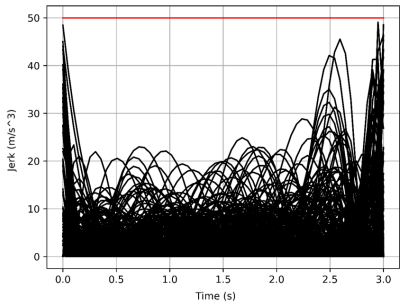

(e) Roll

Fig. 14: The cluttered environment simulation, randomly generated obstacles in $20m \times 20m$ environment with an obstacle density of $> 0.1 obstacles/m^2$.

tion to these results, there were only three scenarios terminated by pre-defined safety rules, demonstrating that the RL agent can not generate a dynamically feasible collision-free trajectory because of the deficiency of enough time to maneuver. Furthermore, another 500 randomly generated scenarios are generated, with increased control point numbers to define a B-spline trajectory. In this case, five scenarios were terminated due to no solution found under dynamical limitations. From a practical point of view, It is possible to foresee these situations and hold the action to avoid possible crash situations.




(a) Velocity (b) Acceleration

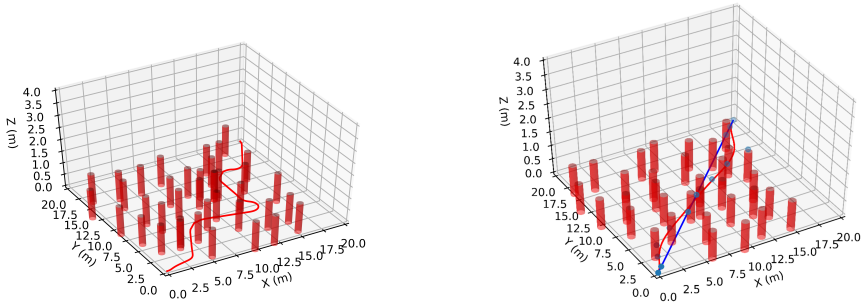



(c) Jerk

Fig. 15: 500 different flight scenarios where all the obstacles are positioned randomly on the map are tested. The result of the flight tests are shown for the following metrics (a) velocity $m/s$, (b) acceleration $m/s^2$,and (c) jerk $m/s^3$.

Table 1: Comparisons with other similar trajectory replanning methodologies

| | | |
|---|---|---|
| Liu et al. [5] | $160ms$ | 3.4 GHz dual-core i7 Intel NUC |
| Burri et al. [14] | $> 40ms$ | AscTec Firefly 2.4 GHz Controller |
| Chen et al. [9] | $> 34ms$ | 3.20 GHz Intel Core i5-4570 CPU |
| Lopez et al. [20] | $\leq 5.06ms$ | 2.70GHz Intel Core i7-2620M |
| CL-RRT* [34] | $\geq 650ms$ | Intel Xeon 2.4GHz |
| Our method | $1.2ms$ | Intel Xeon 2.4GHz |




(a) CL-RRT* solution at 650ms

(b) Our methodology at 1.2 ms

Fig. 16: Comparison with Closed-loop RRT* solution for trajectory replanning

## 5.2 Performance Comparison with Other Algorithms

Considering the real-time applicability, we have measured the computation times, which are conducted on Intel Xeon 2.4 GHz. For the worst case scenario, we have observed that it takes 0.512ms to produce a new location for the control point of the DRL agent. And with the knot insertion method, generating and updating the trajectory takes $1.2ms$. The computation time table with other reference methodologies with their computational power is given below, and it is worth noting that $1.2ms$ computation time starts from after obstacle detection to the end of the trajectory updating. The reported computation times in the table might include the point cloud processing to detect the obstacles in the environment, which typically has utilization around $\%5 - 10$.

Then we have concentrated on the RRT* algorithm, as it is a well-known and well-studied sampling-based trajectory planning methodology assuring asymptotic optimality [34]. Considering the dynamical feasibility, we applied the closed-loop modification of RRT* (CL-RRT*) algorithm to compare with the presented method.

It should be noted that, in the agile collision avoidance problem, the goal is to generate a solution as fast as possible; therefore, we have chosen to use the first feasible trajectory of CL-RRT*. In sampling-based algorithms, while

the number of samples goes to infinity, the algorithm's solution converges to the optimal solution [35]. Hence, they need to use large samples to generate nearly optimal solutions.

When we compare our solution with the CL-RRT* algorithm, while our algorithm provides a re-planned trajectory in 1.2 ms, the CL-RRT* algorithm generates a feasible trajectory after 650 ms. This result is expected, as the CL-RRT* has no ability to use existing (but compromised) flight trajectory, while our methodology exploits it, in addition to offline training. As seen in Fig. 16, the first product of the closed-loop RRT* algorithm, which presents after $50 - 75$ samplings on average, is far from the optimal trajectory. The detailed comparison between the CL-RRT* algorithm with others can be seen in [36].

### 5.3 Hardware Implementation

We have applied the methodology integrating with agile trajectory tracking controller to a small UAV hardware platform flying under motion capture system. A video including explanation of the algorithm and a couple of random scenarios can bee seen in the following link: https://youtu.be/8IiLQFQ3V0E.

The air vehicle used in this work is Crazyflie 2.1 [37]. Fig. 17 shows the architecture used in the implementation. The initial desired B-spine trajectory reference is sent to the trajectory tracking controller by the navigation system. The VICON motion capture system is used at a 100 HZ to obtain the drone and the obstacles positions. The off-board trajectory tracking controller takes the desired trajectory defined as flat outputs and the estimated states of the air vehicle and sends the desired thrust and angles to the on-board attitude controller which runs at 500 Hz.

We used a sensor emulator which is explained in Section-2 as a perception model with $1m$ sensing range and $20°$ field of view. The navigation system works at 100 HZ an Intel Core i7 CPU. It generates the trajectory references for the trajectory tracking controller. If there are any obstacles in the sensor FOV, the collision avoidance algorithm is triggered by the sensor emulator. Then, the algorithm generates collision avoidance maneuver by only relocating newly added control point.

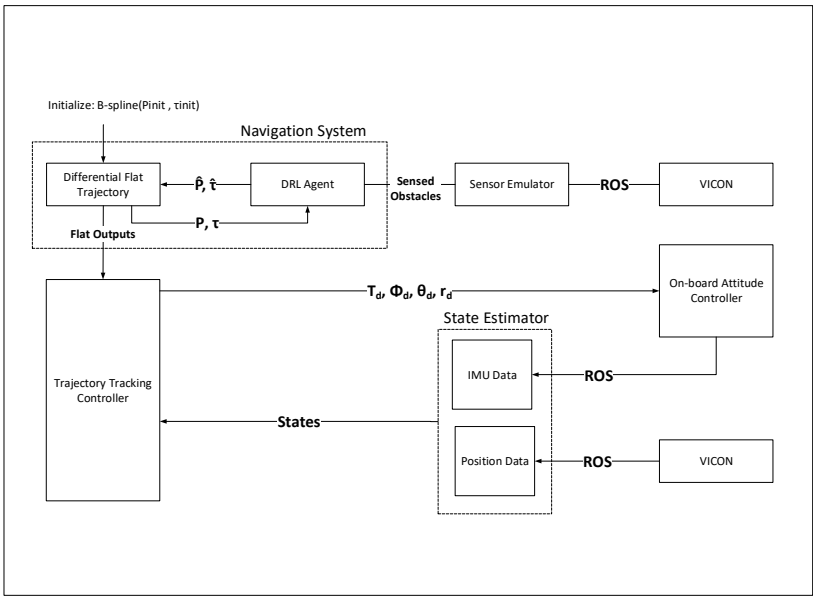



Fig. 17: The system architecture for hardware implementations

## 6 Conclusion

In this paper, we have proposed a fast trajectory replanning methodology based on B-spline regeneration with deep reinforcement learning. Our aim was to simplify and accelerate the replanning problem by providing a rigorous local solution without breaking continuity over the trajectory. We have applied the B-spline theory for local support property and apply it for defining the agile flight trajectory. For regenerating the local trajectory segments, knot insertion methodology with control point reallocation has been used. As the geometric and dynamic form of the trajectory based on the location of the control points and the knot intervals, we have turned the control point reallocation problem into a constrained optimization problem and solved through Deep Reinforcement Learning (DRL). With Principal Proxy Optimization (PPO), we have solved the constrained optimization problem enabling to generate dynamically feasible (considering dynamical constraints) local trajectory segment providing fast collision avoidance. We have trained the DRL agent with different environmental complexities, as we defined through obstacle number per $m^2$. Through the batch simulations, we have shown that the proposed methodology is enabling to solve fast trajectory replanning problem under given or hard dynamical constraints, and providing real-time applicability for such fast collision avoidance applications in agile unmanned aerial vehicles.

As future work, we investigate utilizing adaptive increment in the number of control points regarding the defined complexities such as obstacle density, if known/estimated. In practice, this evaluation might be obtained through the

algorithm's number of trials while searching for a dynamically feasible trajectory, and the knot number might be increased at certain thresholds. Considering the strong functional relationship between the number of control points and the knot vector's dimension, we will have limited control over the number of knots. Yet, the nonuniform knot vector utilization, enabling to morph the focused region of interest's length, will provide additional authority and robustness to generate a replanned trajectory. We will further be benefiting from integrating nonuniform knot vector and nonstationary size of it.

Another natural advancement of the work will be the extension of its implementation to the 3D environments. On the replanning side, the theory behind the proposed methodology is intrinsically generic and independent of the work-space dimension. Obvious requirements will be integrating the control point search over the additional dimensions and applying higher dimensional Euclidian distances. On the implementation side, attitude tracking over the generated trajectory at trustworthy levels will be imperative. We will further enhance our methodology by supporting with precise trajectory tracking for an extension to 3D environment implementations.

Moreover, the proposed methodology is being integrated into the motion capture system and UAV hardware platforms with developed agile trajectory tracking controller. We will be further studied to extend the algorithm to use in the environments with dynamic obstacles through approximate motion estimation.

## References


1. D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.
2. M. Van Nieuwstadt and R. M. Murray, "Real time trajectory generation for differentially flat systems," *IFAC Proceedings Volumes*, vol. 29, no. 1, pp. 2301–2306, 1996.
3. C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," *In Robotics Research*, vol. 8, no. 11, pp. 649–666, 2016.
4. M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 1872–1878.
5. Sikang Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1484–1491.
6. M. Hehn and R. DAndrea, "Quadrocopter trajectory generation and control," *in World Congress*, vol. 18, no. 1, pp. 1485–1491, 2011.
7. C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.
8. M. Watterson and V. Kumar, "Safe receding horizon control foraggressive mav flight with limited range sensing," in *in Intelligent Robots and Systems (IROS)*. 2015 IEEE/RSJ International Conference on. IEEE, 2015, pp. 3235–3240.
9. Jing Chen, Tianbo Liu, and Shaojie Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1476–1483.

10. F. Gao and S. Shen, "Online quadrotor trajectory generation and autonomous navigation on point clouds," in *in Safety, Security, and Rescue Robotics (SSRR)*. IEEE International Symposium on., 2016, pp. 139–146.
11. V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for mavs using uniform b-splines and a 3d circular buffer," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017. [Online]. Available: http://dx.doi.org/10.1109/IROS.2017.8202160
12. S. B. Mehdi, R. Choe, V. Cichella, and N. Hovakimyan, "Collision avoidance through path replanning using bézier curves," in *AIAA Guidance, Navigation, and Control Conference*, 2015, p. 0598.
13. P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.
14. M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 1872–1878.
15. T. M. Howard, C. J. Green, A. Kelly, and D. Ferguson, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," *Journal of Field Robotics*, vol. 25, no. 6-7, pp. 325–345, 2008.
16. M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, Dec 2015.
17. C. J. Green and A. Kelly, "Toward optimal sampling in the space of paths," in *In Proceedings of the International Symposium of Robotics Research*. Citeseer, 2007.
18. S. Daftry, S. Zeng, A. Khan, D. Dey, N. Melik-Barkhudarov, J. A. Bagnell, and M. Hebert, "Robust monocular flight in cluttered outdoor environments," *CoRR*, vol. abs/1604.04779, 2016.
19. M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 2452–2458.
20. B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5759–5765.
21. ——, "Aggressive collision avoidance with limited field-of-view sensing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 1358–1365.
22. S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," *CoRR*, vol. abs/1202.0253, 2012. [Online]. Available: http://arxiv.org/abs/1202.0253
23. M. G. Cox, "The numerical evaluation of b-splines," *IMA Journal of Applied Mathematics*, 1972.
24. C. D. Boor, "On calculating with b-splines," *Journal of Approximation theory*, 1972.
25. L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 2012.
26. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
27. S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, vol. 2, 2002, pp. 267–274.
28. M. Dorn, "Aircraft agility: The science and the opportunities," in *Systems and Operations Conference*. AIAA, 1989.
29. J. Verbeke and J. D. Schutter, "Experimental maneuverability and agility quantification for rotary unmanned aerial vehicle," *International Journal of Micro Air Vehicles*, vol. 10, no. 1, pp. 3–11, 2018. [Online]. Available: https://doi.org/10.1177/1756829317736204
30. W. Herbst, "Agility," in *Briefing Presented at the Workshop on Agility Metrics Held at the AF Flight Test Center*. Edwards AFB, 8-10 Mar 1988.
31. A. Skow, "Transient agility enhancements for tactical aircraft," in *Eidetics International Report (TR89-001) Prapered Under USAF Contracts F33615-85-C-0120 and F33657-87-C-2045 for ASD/XRM*, Jan 1989.

32. ——, "Innovative performance and maneuverability measures of merit for air combat," in *Eidetics International Report (TR-210) USAF Contracts F33615-85-C-0120 for ASD/XRM*, Jan 1989.
33. P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.
34. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406761
35. S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 7681–7687.
36. S. Choudhury, "Adaptive motion planning," Ph.D. dissertation, Pittsburgh, PA, February 2018.
37. "Crazyflie 2.1," https://www.bitcraze.io/crazyflie-2-1/.