Model Predictive Sample-based Motion Planning for Unmanned Aircraft Systems

Model Predictive Sample-based Motion Planning for Unmanned Aircraft Systems

Stefano Primatesta[1], Alessandro Pagliano[2], Giorgio Guglieri[1], Alessandro Rizzo[2]

***Abstract*— This paper presents an innovative kinodynamic motion planning algorithm for Unmanned Aircraft Systems, called MP-RRT$^{\#}$. MP-RRT$^{\#}$ leverages the idea of RRT$^{\#}$ and the Model Predictive Control strategy to solve a motion planning problem under differential constraints. Similar to RRT$^{\#}$, the algorithm explores the map by constructing an asymptotically optimal graph. Each time the graph is extended with a new vertex, a forward simulation is performed with a Model Predictive Control to evaluate the motion between two adjacent vertices and compute the trajectory in the state space and the control space. As result, the MP-RRT$^{\#}$ algorithm generates a feasible trajectory for the UAS satisfying dynamic constraints.**

Preliminary simulation results corroborate the proposed approach, in which the computed trajectory is executed by a simulated drone controlled with the PX4 autopilot.

I. Introduction

In the last years Unmanned Aircraft Systems (UAS) have been widely studied due to their flexibility to provide many applications, such as mapping, surveillance, package delivery, to name a few [1]. Their extensive use has induced the rapid growth of the related research areas, making possible the development of intelligent drones able to fly autonomously [2].

The development of an unmanned aircraft with autonomous flight is very complex [3], requiring the implementation of the essential tasks of autonomous robots: motion planning and control, localization and mapping, and perception [4]. However, despite the level of autonomy of the vehicle, the motion planning problem is omnipresent. In simple words, motion planning aims to find the control input that moves the vehicle from an initial to a target state satisfying some constraints, such as avoiding obstacles and considering the vehicle kinematics and dynamics [5].

Often, the motion planning problem is split into two parts: path planning and path tracking. In [6] the authors propose a potential field approach to compute a path, then followed using a multi-constrained Model Predictive Control. Another two-stage strategy is proposed in [7], in which the Rapidly-exploring Random Tree (RRT) algorithm is used to plan a reference trajectory tracked by a Linear Quadratic Regulation (LQR) controller. Similarly, in [8], the path computed with RRT is post-optimized using a Model Predictive Control to define a feasible trajectory. However, these approaches do not always guarantee the dynamic feasibility of the computed path and, as a consequence, the UAS will take a different trajectory than the planned one.

Kinodynamic motion planning algorithms overcome this problem because they attempt to solve the motion planning problem satisfying, simultaneously, kinematic and dynamics constraints [9]. In the literature there are several studies on kinodynamic motion planning solved using Bezier curves [10], harmonic potential field [11] and a learning approach [12], to name a few.

In the last years, a popular class of algorithms widely used for kinodynamic motion planning is sampling-based planners. Sampling-based algorithms, such as the Rapidly-exploring Random Tree (RRT) and the Probabilistic Roadmap (PRM) are suitable to rapidly find solutions even in high-dimensional spaces [13]. In particular, one of the most popular sampling-based technique is the RRT* algorithm proposed in [14]. RRT* enhances the original RRT algorithm by providing a near optimal solution. As a consequence, many variants of RRT* have been developed to provide anytime planning [15], real-time path planning [16], multi-agent planning [17], and more others [18].

A very interesting RRT-based algorithm, called RRT$^{\#}$ is presented in [19]. RRT$^{\#}$ is asymptotically optimal and ensures that the constructed incremental graph always contains lowest-cost path information for promising vertices, i.e. vertices that can improve the solution [19]. In fact, a drawback of the RRT* algorithm resides in the rewiring procedure, in which any new information in the graph is updated locally, without providing a globally propagation in the graph.

A sampling-based kinodynamic algorithm is proposed for the first time by LaValle and Kuffner in [20], in which RRT constructs a tree of trajectories in the state space generated by sampling the control input of the vehicle and, then, simulating its motion. As a consequence, the computed trajectory satisfies the vehicle dynamics and can be easily executed by the vehicle. Based on this algorithm, several approaches have been developed in the last years [21], [22]. However, sampling in the control space is not efficient for vehicles with complex dynamics, because often a sample configuration can result in an infeasible trajectory and, as a consequence, many iterations are required to compute a good quality solution. To solve this problem, in [23] the *closed-loop* RRT (CL-RRT) is proposed, in which vertices are sampled in the reference space and a closed-loop prediction computes a trajectory in the state space. The same approach is also used

[1]S. Primatesta and G. Guglieri are with the Department of Mechanical and Aerospace Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy. Corresponding author: Stefano Primatesta (e-mail: stefano.primatesta@polito.it)

[2]A. Pagliano and A. Rizzo are with the Department of Electronics and Telecommunications, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy.

in [24] applied with the RRT$^{\#}$ algorithm. Similarly, in [25] a kinodynamic RRT* is developed using Dubins curves and the double integrator, whereas a LQR controller is used in [26].

This paper presents a kinodynamic motion planning algorithm called MP-RRT$^{\#}$, leveraging the ideas of the RRT$^{\#}$ algorithm [19] and the Model Predictive Control [27]. Similar to the concept proposed in [23], the algorithm samples in the reference space and, then, a Model Predictive Control is used to provide a forward simulation to evaluate the motion between vertices and to compute a trajectory in the state space and in the control space. Specifically, the MP-RRT$^{\#}$ is used to solve the motion planning problem for a multicopter. As result, the MP-RRT$^{\#}$ returns a feasible trajectory for the UAS satisfying dynamic constraints.

The use of the MPC in the graph construction of a RRT-based algorithm is a novelty and introduces some benefits, such as a the use of a model-based control strategy, a small prediction error and a generation of a feasible trajectory easily executable by the vehicle.

The rest of the paper is organized as follows. Section II defines the optimal motion planning problem. The proposed algorithm is described in Section III, detailing the pseudocode, the UAS model and the Model Predictive Control strategy adopted. Section IV describes the experimental result and our conclusions are drawn in Section V.

II. Problem Formulation

This section defines the motion planning problem assumed in this work. Let's define the UAS dynamic model

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \tag{1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ is the state of the system of dimension n_x, and $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ is the control input of dimension n_u. Both states and control inputs should respect the following constraints

$$\boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}, \tag{2}$$

$$\boldsymbol{u}(t) \in \mathcal{U}. \tag{3}$$

The Equation (2) defines a constraint to avoid obstacles by setting the vehicle state in the *free state space* $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$, with $\mathcal{X}$ the state space, and $\mathcal{X}_{\text{obs}}$ the space occupied by obstacles. The Equation (3) defines the bounds of the control input.

Given the initial state of the UAS $\boldsymbol{x}(0) = \boldsymbol{x}_0$ at time $t = 0$ and the desired target state defined by the goal region $\mathcal{X}_{\text{goal}} \subset \mathbb{R}^{n_x}$. The aim of the motion planning problem is defining an optimal state trajectory $\overline{\boldsymbol{x}}^* : [0, t_f] \in \mathcal{X}_{free}$ and an optimal control input sequence $\overline{\boldsymbol{u}}^* : [0, t_f] \in \mathcal{U}$ over a finite horizon from 0 to t_f for moving from the initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ to a final state in the goal region $\boldsymbol{x}(t_f) \in \mathcal{X}_{\text{goal}}$. $\overline{\boldsymbol{x}}^*$ and $\overline{\boldsymbol{u}}^*$ are computed by minimizing a cost function $\text{Cost}(\cdot)$ and satisfying constraints (2) and (3). Hence, the optimal motion is the solution of the following program

$$\begin{aligned} & \overline{\boldsymbol{x}}^*, \overline{\boldsymbol{u}}^* = \arg\min \ \text{Cost}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \text{subject to} \quad & \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ & \boldsymbol{x}(t_f) = \boldsymbol{x}_{\text{goal}} \in \mathcal{X}_{\text{goal}} \\ & \forall t \in [0, t_f], \ \boldsymbol{x}(t) \in \mathcal{X}_{\text{free}} \\ & \forall t \in [0, t_f], \ \boldsymbol{u}(t) \in \mathcal{U}. \end{aligned} \tag{4}$$

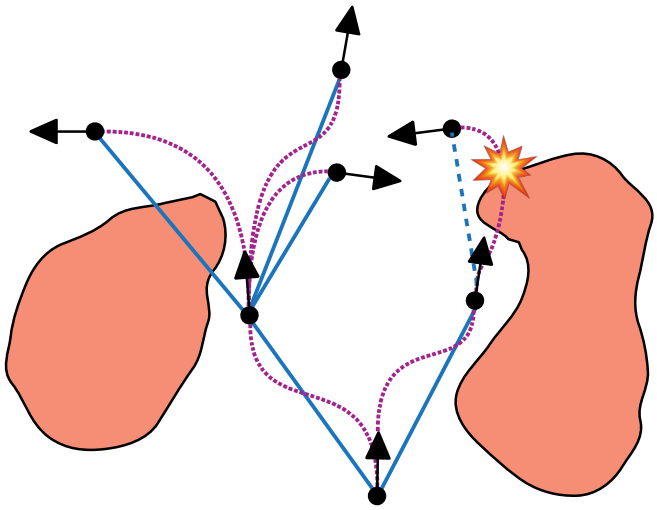


Figure 1. Example of the graph constructed with MP-RRT$^{\#}$. The graph consists in vertices (in black) and edges (in blue). Edges are evaluated by computing the trajectories (in magenta) with the Model Predictive Control. An edge is not valid if the corresponding trajectory hits an obstacle.

III. The MP-RRT$^{\#}$ strategy

This section introduces the MP-RRT$^{\#}$ (Model Predictive Rapidly-exploring Random Tree "sharp") algorithm, which extends the RRT$^{\#}$ algorithm [19] with the Model Predictive Control philosophy to compute a near optimal trajectory for UAS.

As other kinodynamic RRT-based algorithms, MP-RRT$^{\#}$ generates an incremental graph of feasible trajectories to explore the search space and to reach a specific target state. Unlike existing works [20], [21], [22] that search for an optimal trajectory by randomly sample in the control input space, our approach samples an input of the closed-loop system as proposed in [23], i.e. in the reference space.

Generally, sampling the control input is inefficient. Most of the sampled inputs are discarded because they provide a bad behavior of the system. Hence, especially when the dynamics are complex, a lot of iterations are required to construct the graph and compute an optimal trajectory. On the contrary, our approach samples an input of the closed-loop system, i.e. a reference $r(t) \in \mathbb{R}^{n_r}$. Then, the algorithm performs a forward simulation using a Model Predictive Control strategy computing a state trajectory and the optimal control input to follow the sampled reference. Thanks to the MPC logic, the computed trajectory satisfies the constraint of Equation (2). Anyway, the feasibility of the computed trajectory is checked later verifying if it collides with obstacles and, then, satisfying the Equation (3). In our strategy the

MPC does not avoid obstacles. In fact, unfeasible trajectories are simply discarded, as well as the edge corresponding to the motion. This choice is made to avoid a too complex controller that is called several times during the proposed motion planning.

This strategy is more efficient than other kinodynamic RRT-based approaches [20], [21], [22] because each sampled reference state corresponds to a feasible trajectory providing an optimal motion. This requires fewer samples to construct an exploration tree and, then, to compute an optimal trajectory. Moreover, sampling a reference state is also more efficient than directly sampling the vehicle state. In fact, for vehicles with complex dynamics, the reference command has generally a lower dimension than the dimension of the vehicle state and, then, $n_r \ll n_x$. For instance, in our work, the UAS state is defined with 8 variables, while the reference with only 3 variables.

The above mentioned logic is used to generate a graph of feasible trajectories using the RRT$^{\#}$ algorithm. Figure 1 shows a simple example of the proposed strategy. Note that if a trajectory enters in the obstacle space, the related edge is not included in the graph. On the contrary, even if the edge crosses an obstacle, it is not discarded if the related trajectory does not collide with obstacles.

A. Algorithm

The proposed algorithm is based on the RRT$^{\#}$ algorithm proposed in [19]. RRT$^{\#}$ is a variant of the Rapidly-exploring Random Graph (RRG) that ensures a globally optimal graph in the search space.

The main pseudocode of MP-RRT$^{\#}$ is defined in Algorithm 1. The inputs of the algorithm are the initial state x_0, the goal region $\mathcal{X}_{goal}$ and the state space $\mathcal{X}$ in which the motion planning searches for a solution. First, the set of vertices V, the set of edges E and the graph $\mathcal{G}$ are initialized (from lines 2 to 4). Then, the iterative procedure of the construction of the graph starts and continues until a certain number N of vertices are sampled and added to the graph (lines 5 to 8). Specifically, a vertex r_{rand} is randomly sampled (line 6) and the graph is extended by adding the new vertex (line 7). The $\mathrm{Replan}()$ function propagates this update on the graph (line 8). Both the $\mathrm{Extend}()$ and $\mathrm{Replan}()$ functions are detailed in Algorithms 2 and 4, respectively. Finally, the branch $\mathcal{T}$ connecting the initial and the target states is extracted from the graph (line 9) and returned as the solution of the algorithm.

With a slight abuse of notation, with vertex r we denote a reference state in the state space $\mathcal{X}$. Then, the reference state is used to compute the resulting state trajectory $\overline{x}$ to move toward r.

The Extend procedure is a crucial element for the proposed approach because it expands the graph by adding a new vertex and, then, by computing the related state trajectory using the Model Predictive Control. Specifically, this procedure is detailed in Algorithm 2. First, the vertex r is connected to the nearest vertex r_{nearest} (line 4). The $\mathrm{Nearest}()$ function selects the vertex with the minimum Euclidean distance from

Algorithm 1: The MP-RRT$^{\#}$ algorithm

```
1  MP-RRT# (x_0, X_goal, X)
2      V ← {x_0};
3      E ← ∅;
4      G ← (V, E)
5      for i = 0 to N do
6          r_rand ← Sample();
7          G ← Extend(G, r_rand);
8          Replan(G);
9      T ← SpanningTree(G);
10     return T
```

r. The $\mathrm{ComputeTrajectory}()$ function computes the optimal state trajectory $\overline{\mathrm{x}}$ of moving from r_{nearest} to r (line 5). Then, if the computed trajectory is valid, i.e. it does not collide with obstacles, the *cost-to-come* of vertex r, denoted by $g(r)$ is computed by increasing the cost at the previous vertex with the cost of the trajectory $\overline{x}$, denoted by $c(\overline{x})$ (line 7). In line 8 all the neighbor vertices of r are added to the neighbor set $\mathcal{N}$ and, then, the vertex r is included in the neighbor set of its neighbors (lines 9 to 11). The $\mathrm{Near}()$ function selects all the vertices within a certain radius as defined in [14].

Then, the $\mathrm{FindParent}()$ function searches if one of the neighbors can be the parent vertex of r (line 12), i.e. the neighbor vertex that provides the minimum cost-to-come $g(r)$. The $\mathrm{FindParent}()$ procedure is detailed in Algorithm 3, in which the trajectory between the vertex r and each neighbor vertex $r_{\mathrm{near}} \in \mathcal{N}(r)$ is evaluated seeking for the best parent vertex.

Then, the vertex r is added to the graph and it is included in the priority queue q (line 15).

The priority queue has a crucial role in the RRT$^{\#}$ algorithm [19] because it is a queue of vertices that is evaluated in the $\mathrm{Replan}()$ procedure to propagate any update on the graph. Vertices of the queue are ordered based on their cost $f(r)$ from the highest to the lowest. Specifically, the cost $f(r)$ is the estimated cost to reach the goal passing through the vertex r, inspired by the well-known cost function define in the A* algorithm [28]

$$f(r) = g(r) + \hat{h}(r). \tag{5}$$

$g(r)$ is the *costo-to-come* at the vertex r, i.e. the cost of moving between the starting vertex x_0 and r, with $g(x_0) = 0$. $\hat{h}(r)$ is the estimated cost-to-go to reach the goal state, with $\hat{h}(r_{\mathrm{goal}}) = 0$.

In particular, the $\mathrm{Replan}()$ procedure is detailed in Algorithm 4. This procedure is based on an iterative loop that updates only promising vertices (lines 3 to 13), i.e. vertices that can improve the current solution in the graph. Specifically, the set of promising vertices $V_{\mathrm{prom}} \subset V$ contains vertices inside the relevant region $\mathcal{X}_{\mathrm{rel}}$

$$\mathcal{X}_{\mathrm{rel}} = \{r \in \mathcal{X}_{\mathrm{free}} : f(r) < g(r^*_{\mathrm{goal}})\}, \tag{6}$$

with r^*_{goal} is the vertex in the goal region with the minimum

Algorithm 2: The Extend procedure

```
1  Extend (G, r)
2      (V, E) ← G;
3      E' ← ∅;
4      r_nearest ← Nearest(G, r);
5      x̄ ← ComputeTrajectory(r_nearest, r);
6      if isTrajectoryValid(x̄) then
7          g(r) ← g(r_nearest) + c(x̄);
8      N(r) ← Near(G, r);
9      foreach r_near ∈ N(r) do
10         N(r_near) ← N(r_near) ∪ {r};
11         E' ← E' ∪ {(r_near, r), (r, r_near)};
12     FindParent(r);
13     V ← V ∪ {r};
14     E ← E ∪ E';
15     UpdateQueue(r);
16     return G ← (V, E)
```

Algorithm 3: The FindParent procedure

```
1  FindParent (r)
2      foreach r_near ∈ N(r) do
3          x̄ ← ComputeTrajectory(r_near, r);
4          if isTrajectoryValid(x̄) then
5              if g(r_near) + c(x̄) < g(r) then
6                  g(r) = g(r) + c(x̄);
7                  P(r) = r_near;
```

cost-to-come. Notably, the heuristic cost $\hat{h}(r)$ used to compute $f(r)$ must be admissible, i.e. it should not overestimate the cost-to-go, discarding vertices that would lead to the optimal solution. The evaluation of promising vertices is essential to avoid the propagation toward vertices that can not improve the current solution, speeding up the algorithm.

The first element of the queue is selected (line 4) and removed from q (line 5). Then, the procedure verifies if the current vertex can improve the cost-to-come of its neighbors (lines 6 to 13) as a new parent vertex. This is verified by computing the cost-to-come of the resulting state trajectory of moving from r to the neighbor vertex r_{nbh}. Exactly as in Algorithm 3, line 9 checks if r_{nbh} is a promising vertex and, in line 10, if r can be the new parent vertex of r_{nbh}. If occurs, the vertex r_{nbh} is included in q to be evaluated in the Replan() procedure.

B. UAS model

Before explaining the Model Prediction Control strategy adopted, we detailed the model of the Unmanned aircraft assumed in this work. In particular, we consider a multicopter with a linear model approximated around its hovering conditions [29], where small attitude angle variations are assumed and the vehicle heading is aligned with the inertial frame x-axis, i.e. $\psi = 0$. Hence, the linear model of the system is

Algorithm 4: The Replan procedure

```
1  Replan (G)
2      (V, E) ← G;
3      while f(q.top()) ≺ g(r*_goal) do
4          r = q.top();
5          q.pop();
6          foreach r_nbh ∈ N(r) do
7              x̄ ← ComputeTrajectory(r, r_nbh);
8              if isTrajectoryValid(x̄) then
9                  if g(r) + c(x̄) + ĥ(r_nbh) < g(r*_goal)
                   then
10                     if g(r) + c(x̄) < g(r_nbh) then
11                         g(r_nbh) = g(r) + r(x̄);
12                         P(r_nbh) = r;
13                         UpdateQueue(r_nbh);
```

defined as follows

$$\dot{x}(t) = A_c x(t) + B_c u(t), \tag{7}$$

with A_c is the state matrix in continuous time

$$A_c = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -a_x & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & -a_y & 0 & 0 & -g \\ 0 & 0 & 0 & 0 & 0 & -a_z & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} \end{bmatrix}, \tag{8}$$

and B_c is the input matrix in continuous time

$$B_c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{k_\phi}{\tau_\phi} & 0 & 0 \\ 0 & \frac{k_\theta}{\tau_\theta} & 0 \end{bmatrix}, \tag{9}$$

where a_x, a_y and a_z are the drag coefficients, g is the gravity acceleration, τ_ϕ is the roll time constant, τ_θ is the pitch time constant, k_ϕ is the roll gain and k_θ is the pitch gain. The state vector is $x = \begin{bmatrix} p^T & v^T & {}^{\mathbb{W}}\phi & {}^{\mathbb{W}}\theta \end{bmatrix}^T$, where p is the position vector of the UAS in the three-dimensional space, v is the velocity vector, ${}^{\mathbb{W}}\phi$ and ${}^{\mathbb{W}}\theta$ are the roll and pitch angles in the inertial frame $\mathbb{W}$. The input vector is $u = \begin{bmatrix} {}^{\mathbb{W}}\phi_d & {}^{\mathbb{W}}\theta_d & T \end{bmatrix}^T$, where ${}^{\mathbb{W}}\phi_d$ and ${}^{\mathbb{W}}\theta_d$ are the roll and pitch control command in the inertial frame and T is the thrust control command.

The controller is implemented in the discrete time and, as a consequence, the UAS model is discretized as follow

$$A = e^{A_c T_s}, \tag{10}$$

$$B = \int_0^{T_s} e^{A_c d\tau} d\tau B_c, \tag{11}$$

where T_s is the sampling time. In this work we used a simple and linear model to avoid a too complex optimization for the MPC.

C. Model Predictive Control

The MP-RRT$^{\#}$ algorithm uses a Model Predictive Control to compute the optimal state trajectory of moving between two adjacent vertices and, then, to evaluate the cost of the trajectory.

Based on the UAS model previously defined, in this work we implement a Linear Model Predictive Control inspired by [29], where the authors propose a MPC-based trajectory tracking.

Specifically, the MPC searches for an optimal trajectory by optimizing the following cost function J

$$\begin{aligned} &J(\overline{x}, \overline{u}) = \\ &\Big(\sum_{k=0}^{H_p-1} (x_k - x_{ref,k})^T Q_x (x_k - x_{ref,k}) \\ &+ (u_k - u_{k-1})^T R_\Delta (u_k - u_{k-1}) \Big) \\ &+ (x_{H_p} - x_{ref,H_p})^T Q_{final} (x_{H_p} - x_{ref,H_p}), \end{aligned} \tag{12}$$

with H_p is the prediction horizon, $\overline{u}$ is the input vector with $\overline{u} = \begin{bmatrix} u_0 & u_1 & \dots & u_{H_p} \end{bmatrix}^T$ and $u_k \in \mathbb{R}^3$ for k from 0 to $H_p - 1$. $\overline{x}$ is the vector state $\overline{x} = \begin{bmatrix} x_0 & x_1 & \dots & x_{H_p} \end{bmatrix}^T$ with $x_k \in \mathbb{R}^8$ for k from 0 to H_p. $\overline{x}_{\text{ref}}$ is the vector reference state $\overline{x}_{\text{ref}} = \begin{bmatrix} x_{\text{ref,0}} & x_{\text{ref,1}} & \dots & x_{\text{ref,H}_\text{p}} \end{bmatrix}^T$ with $x_{\text{ref,k}} \in \mathbb{R}^8$ for k from 0 to H_p. Q_x is the penalty matrix on the state error, R_Δ is a penalty matrix on the variation on the control input, and Q_{final} is the terminal cost matrix on the last state error. The computation of Q_{final} is done by solving the Algebraic Riccati Equation iteratively [30]. Q_x, R_Δ, Q_{final} are positive semidefinite matrices.

Hence, the following convex optimization problem is solved

$$\begin{aligned} \overline{x}^*, \overline{u}^* = \min_{U,X} \quad & J(\overline{x}, \overline{u}) && (13) \\ \text{subject to} \quad & x_{k+1} = Ax_k + Bu_k && (14) \\ & u_k \in \mathcal{U} && (15) \\ & x_0 = x(t_0) && (16) \end{aligned}$$

This MPC problem is solved each time the MP-RRT$^{\#}$ algorithm evaluates a motion of moving between two adjacent vertices. Anyway, in order to perform the optimization problem of Equations (13), we define a reference trajectory x_{ref} connecting two vertices using Dubins curves, a suitable solution to achieve flyable path, due to their simplicity and performances.

Dubins curves are introduced in [31] and they refer to the shortest path between two poses in the two-dimensional space considering a constant radius curvature. This solution fits with the proposed work, since the algorithm is implemented in the two-dimensional space. However, Dubins curves are also extended to the three-dimensional space [32] and with a variable radius curvature [33].

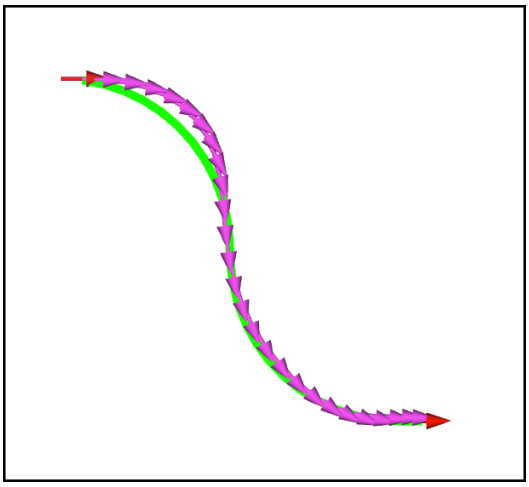


Figure 2. Example of reference trajectory computed using Dubins curves and connecting two adjacent vertices. The green line is the reference trajectory, whereas magenta arrows are the state trajectory computed using the Model Predictive Control.

Given the two-dimensional pose of the aircraft $\begin{bmatrix} p_x & p_y & p_\beta \end{bmatrix}$ and assuming a constant speed, the differential equation of Dubins curves are:

$$\dot{p_x} = cos(p_\beta) \tag{17}$$

$$\dot{p_y} = sin(p_\beta) \tag{18}$$

$$\dot{p_\beta} = u_c \tag{19}$$

where u_c is normalized in the range between -1 and 1, considering the maximum curvature of the aircraft. The shortest path between two poses can be expressed as a combination of no more than three motion primitives [31]. Hence, only three values of u_c are defined $u_c \in \{-1, 0, 1\}$. The value $u_c = 0$ describes a straight motion (S), $u_c = -1$ the right (R) turn, while $u_c = 1$ the left (L) turn. As a consequence, only six combination of curves exist:

$$\{LRL, RLR, LSL, LSR, RSL, RSR\} \tag{20}$$

Hence, in accordance with MPC philosophy, the optimization problem is solved to follow the reference trajectory. However, only the first control input is applied and the optimization is solved iteratively. Figure 2 shows an example of reference trajectory using Dubins curves followed by the Model Predictive Control strategy.

D. Implementation details

In this Section some implementation details of the MP-RRT$^{\#}$ are exposed. The MP-RRT$^{\#}$ algorithm is implemented considering a two-dimensional space, i.e. flight at fixed altitude. Specifically, the Special Euclidean Group $SE(2)$ is used, in which each admissible configuration is a pose in the two-dimensional space free to translate and rotate. Hence, each state sampled by the algorithm consists of three parameters, i.e. the position of the UAS and an orientation corresponding to the flight direction. Hence, each

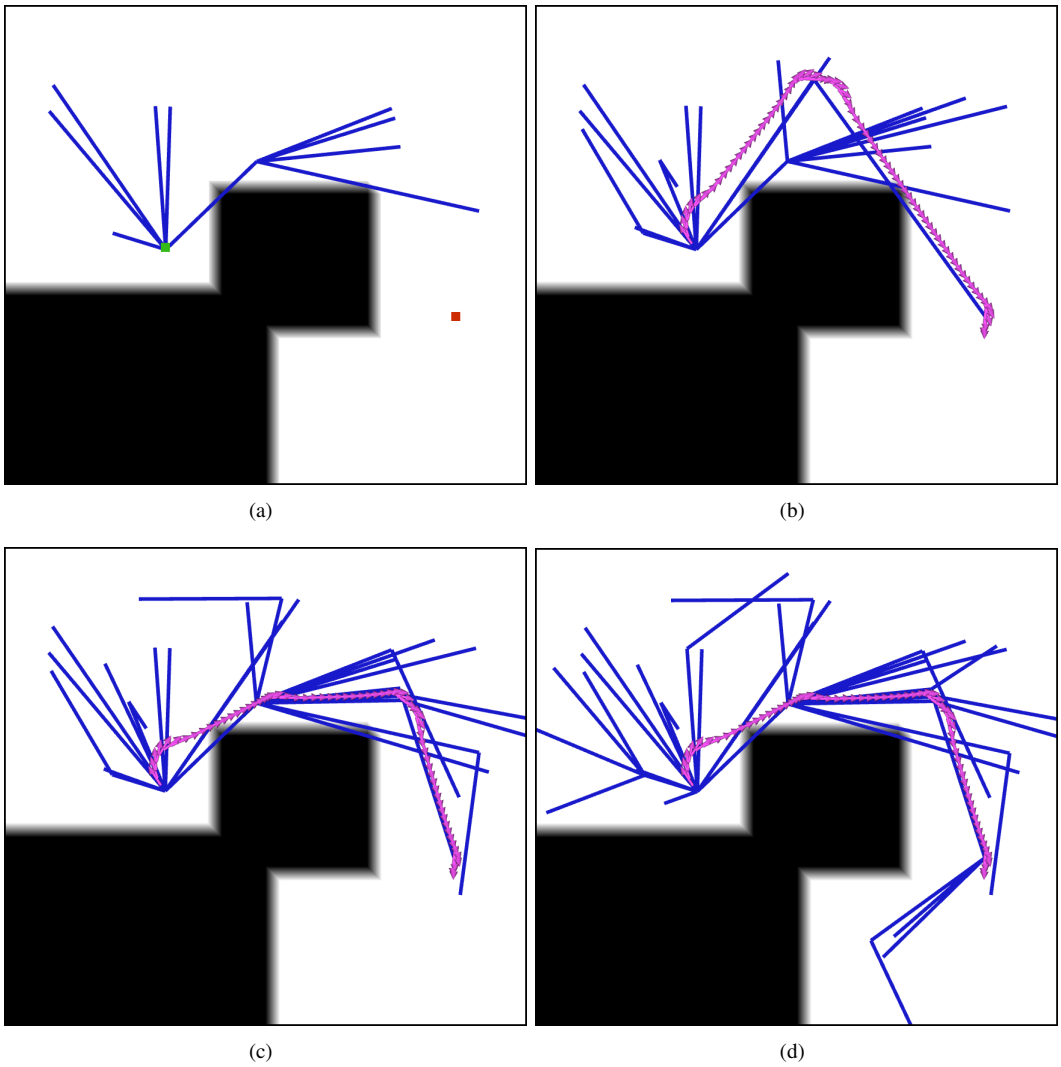


Figure 3. Example of the exploration of the map using the MP-RRT$^{\#}$ algorithm. In (a), the graph with 10 nodes rooted from the start pose, in which the graph does not reach the target pose. In (b), the graph with 20 nodes and with a preliminary solution. In (c), the graph with 30 nodes in which a better solution is found. In (d), the graph is extended with 10 more nodes (40 in total), but the best solution is the graph does not change.

time the MP-RRT$^{\#}$ algorithm evaluates the motion between two states, a reference trajectory is computed using Dubins curves and the MPC computes the optimal state trajectory and control input to follow it.

Then, the motion-cost of the trajectory is computed as follows

$$c(\overline{x}, \overline{u}) = \sum_{i=1}^{M} \|x_{\mathrm{i}} - x_{\mathrm{i-1}}\|_2 + (u_i - u_{i-1})^T R_{\Delta} (u_i - u_{i-1}), \tag{21}$$

with $x_{\mathrm{i}} \in \overline{x}$, $u_{\mathrm{i}} \in \overline{u}$, and M is the size of the trajectory. In particular, the first term evaluates the Euclidean distance of the trajectory, and the second term evaluates the variation of the control input exactly as in Equation (12).

On the other hand, the cost-to-go $\hat{h}(r)$ is computed as the distance of the Dubins curve between the vertex r and the goal region $\mathcal{X}_{\mathrm{ref}}$.

IV. Results

The proposed strategy is implemented in C++ using the Robot Operating System (ROS) [34] framework. Specifically, the MP-RRT$^{\#}$ algorithm is implemented using the Open Motion Planning Library (OMPL) [35] that consists in many state-of-the-art sampling-based algorithms and offers many functionalities to facilitate the implementations of new algorithms. The optimization of the Model Predictive Control is solved using CVXGEN [36], a tool for code generation for convex optimization. CVXGEN can be used to generate fast custom code for small, QP-representable convex optimization problems. The mathematical problem is translated into a high speed solver that is twelve to thousand-times faster than other popular optimizers [36]. Hence, the linear model of the UAS and the Linear MPC problem of Equations (12) and (13) are included and solved with CVXGEN.

The experimental tests are performed considering the multicopter Asctec Firefly and using the parameters listed

Parameter	Value
a_x	0.01
a_y	0.01
a_z	0
k_ϕ	0.9
k_θ	0.9
τ_ϕ	0.250 s
τ_θ	0.255 s

TABLE I

PARAMETERS USED FOR THE UAS MODEL.

in Table I.

The MP-RRT$^\#$ is executed considering a maximum cruise velocity of $2.5m/s$ and the reference trajectory is computed with Dubins curves with a curvature radius of 2 m. The admissible control input is defined with the following constrains

$$-0.436 \text{ rad} \leq^{\mathbb{W}} \phi_d \leq 0.436 \text{ rad} \quad (22)$$

$$-0.436 \text{ rad} \leq^{\mathbb{W}} \theta_d \leq 0.436 \text{ rad} \quad (23)$$

$$-4.80 \text{ N} \leq T \leq 10.19 \text{ N} \quad (24)$$

Figure 2 shows an example of reference trajectory computed with Dubins curves and connecting two vertices. The trajectory is followed by the Model Predictive Control that reaches the target vertex.

The proposed MP-RRT$^\#$ algorithm is tested in different maps to evaluate its effectiveness in computing UAS trajectories. Figure 3 shows the construction of the graph, in which, in Figure 3(a), the graph with 10 nodes does not explore enough the map and does not reach the target pose. In Figure 3(b) the graph is extended with 10 more nodes finding an initial solution. In Figure 3(c), the algorithm finds a new better solution. Finally, in Figure 3(d) the best solution does not change, even if the graph has 10 more nodes (40 in total).

Other tests in more complex maps are shown in Figures 4 and 5. In Figure 4 the trajectory is computed by exploring the map with a graph of 100 nodes.

Similarly, in Figure 5(a), the MP-RRT$^\#$ algorithm explores the map with a graph of 100 nodes computing a solution. The trajectory computed in Figure 5(a) is also executed in a realistic simulation performed using Gazebo and SITL frameworks. Gazebo is an open-source multi-robot simulator fully compatible with ROS [37] able to simulate robots, sensors, and rigid body dynamics. SITL (Software In The Loop) [38] is a software to execute an autopilot on a computer, without using a specific and dedicated hardware. In this work, the simulation uses the PX4 autopilot [39], an open-source flight control software for drones and other autonomous vehicles.

In particular, the state trajectory computed with MP-RRT$^\#$ is uploaded on the PX4 autopilot and, then, executed as shown in Figure 5(b). Even if the environment of Figure 5(b) does not correspond with the map of Figure 5(a), the executed trajectory is the same.

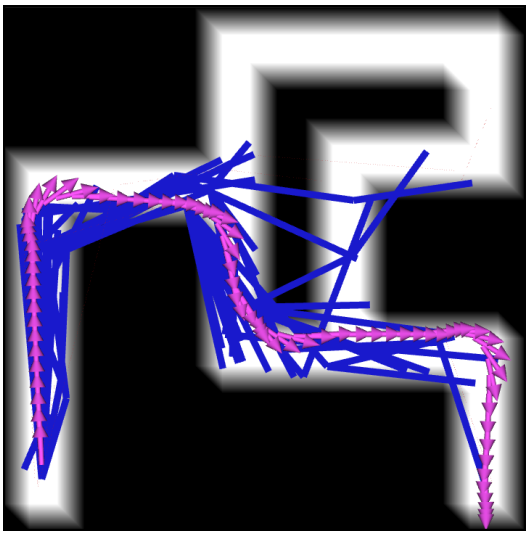


Figure 4. Trajectory computed with the MP-RRT$^\#$ algorithm by constructing a graph of 50 nodes.

V. CONCLUSIONS

In this paper we have presented a model predictive sample-based motion planning for UAS. Specifically, we have introduced a novel algorithm called MP-RRT$^\#$, which extends RRT$^\#$ with the Model Predictive Control philosophy to compute an optimal trajectory for UAS.

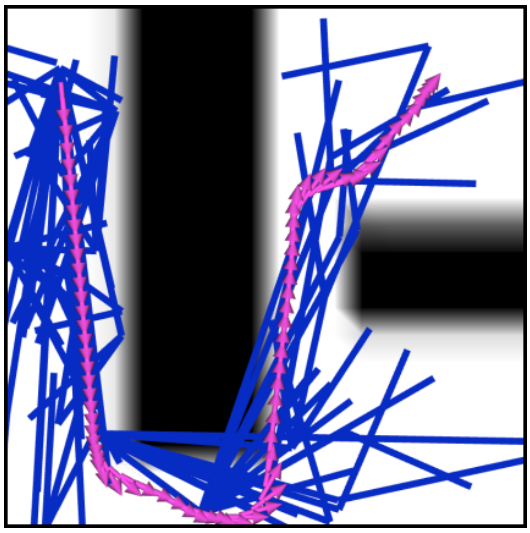
Similarly to RRT$^\#$ the algorithm constructs an asymptotically optimal graph to explore the map and to search for a near optimal solution. Moreover, the Model Predictive Control is used to compute the optimal trajectory of moving between vertices of the graph to evaluate the feasibility of the edge and its cost. As a consequence, the MP-RRT$^\#$ algorithm computes an near optimal trajectory considering the UAS dynamics.

This method differs from other kinodynamic RRT-based algorithms, because MP-RRT$^\#$ samples the input reference of the closed loop system instead of directly sample the control input. This implies a more efficient strategy for vehicles with complex dynamics.

The experimental tests demonstrate how the proposed algorithm is able to compute a good quality trajectory even in complex maps. Moreover, the computed trajectory is executable by a UAS equipped with a professional autopilot.

Even if the proposed algorithm is tested in a simplified scenario, i.e. in a two-dimensional space and using a simple and linear model of the UAS, the MP-RRT$^\#$ algorithm can be adapted to be used in more complex scenarios while increasing the complexity of the algorithm. Moreover, even if this work focuses on UAS, the proposed planning strategy can be used for other systems, such as ground robots, autonomous cars and underwater vehicles.

Future works will include the adaptation of a three-dimensional environment. Moreover, we will exploit the proposed strategy to implement a real-time trajectory planner similar to the work done in [23], [24]. Moreover, experimental tests will be conducted on a real robotic platform.



(a)

(b)

Figure 5. In (a), the trajectory computed with the MP-RRT$^{\#}$ algorithm by constructing a graph of 100 nodes. In (b) the computed trajectory is executed by the PX4 autopilot in a simulation.

ACKNOWLEDGMENT

This work is partially supported by Compagnia di San Paolo and by an Amazon Research Award granted to Dr. A. Rizzo.

REFERENCES

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *Ieee Access*, vol. 7, pp. 48 572–48 634, 2019.

[2] N. Bloise, S. Primatesta, R. Antonini, G. P. Fici, M. Gaspardone, G. Guglieri, and A. Rizzo, "A survey of unmanned aircraft system technologies to enable safe operations in urban areas," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2019, pp. 433–442.

[3] S. Tang and V. Kumar, "Autonomous flight," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 29–52, 2018.

[4] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[5] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[6] J. Ji, A. Khajepour, W. W. Melek, and Y. Huang, "Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 952–964, 2016.

[7] Y. Chen, H. Peng, and J. W. Grizzle, "Fast trajectory planning and robust trajectory tracking for pedestrian avoidance," *Ieee Access*, vol. 5, pp. 9304–9317, 2017.

[8] P. Lin, S. Chen, and C. Liu, "Model predictive control-based trajectory planning for quadrotors with state and input constraints," in *2016 16th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2016, pp. 1618–1623.

[9] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[10] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 2427–2433.

[11] A. A. Masoud, "Kinodynamic motion planning," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, pp. 85–99, 2010.

[12] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *arXiv preprint arXiv:2101.06798*, 2021.

[13] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[14] 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.

[15] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1478–1483.

[16] K. Naderi, J. Rajamäki, and P. Hämäläinen, "Rt-rrt* a real-time path planning algorithm based on rrt," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, 2015, pp. 113–118.

[17] M. Čáp, P. Novák, J. Vokřínek, and M. Pěchouček, "Multi-agent rrt*: Sampling-based cooperative pathfinding," *arXiv preprint arXiv:1302.2828*, 2013.

[18] I. Noreen, A. Khan, Z. Habib *et al.*, "Optimal path planning using rrt* based approaches: a survey and future directions," *Int. J. Adv. Comput. Sci. Appl*, vol. 7, no. 11, pp. 97–107, 2016.

[19] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2421–2428.

[20] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[21] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of guidance, control, and dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[22] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[23] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on control systems technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[24] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4991–4996.

[25] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 7681–7687.

[26] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2537–2542.

[27] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[28] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the

heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
[29] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.
[30] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
[31] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.
[32] T. McLain, R. W. Beard, and M. Owen, "Implementing dubins airplane paths on fixed-wing uavs," 2014.
[33] K. D. Hansen and A. la Cour-Harbo, "Waypoint planning with dubins curves using genetic algorithms," in *2016 European Control Conference (ECC)*. IEEE, 2016, pp. 2240–2246.
[34] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, 2009, p. 5.
[35] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.
[36] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.
[37] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator." in *IROS*, vol. 4. Citeseer, 2004, pp. 2149–2154.
[38] SITL contributors, "SITL guide," http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, 2020.
[39] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 6235–6240.