MP-RRT#: a Model Predictive Sampling-based motion planning algorithm for Unmanned Aircraft Systems

Journal of Intelligent & Robotic Systems (2021) 103:59
https://doi.org/10.1007/s10846-021-01501-3

REGULAR PAPER

MP-RRT#: a Model Predictive Sampling-based Motion Planning Algorithm for Unmanned Aircraft Systems

Stefano Primatesta[1] · Abdalla Osman[2] · Alessandro Rizzo[2]

Received: 20 June 2021 / Accepted: 9 September 2021

Abstract

This paper introduces a kinodynamic motion planning algorithm for Unmanned Aircraft Systems (UAS), called MP-RRT#. MP-RRT# joins the potentialities of RRT# with a strategy based on Model Predictive Control to efficiently solve motion planning problems under differential constraints. Similar to other RRT-based algorithms, MP-RRT# explores the map constructing an asymptotically optimal graph. In each iteration the graph is extended with a new vertex in the reference state of the UAS. Then, a forward simulation is performed using a Model Predictive Control strategy to evaluate the motion between two adjacent vertices, and a trajectory in the state space is computed. As a result, the MP-RRT# algorithm eventually generates a feasible trajectory for the UAS satisfying dynamic constraints. Simulation results obtained with a simulated drone controlled with the PX4 autopilot corroborate the validity of the MP-RRT# approach.

Keywords Unmanned aerial vehicles · Unmanned aircraft · Kinodynamic motion planning · Sampling-based motion planning · Model predictive control

1 Introduction

The use of Unmanned Aircraft Systems (UAS) has increased progressively across a wide range of applications such as remote sensing, search and rescue, security and surveillance, precision agriculture, infrastructure inspection and urban planning, to name a few [35]. Such extensive use of UAS triggered the rapid growth of various related research topics, of which autonomous flight has been driving great interest [3]. The great availability of powerful and efficient computational capabilities has enabled the extensive use of optimal control strategies and machine learning to address the problem of autonomy. However, autonomous flight remains a very challenging task [39], whose solution extends across diverse aspects: perception, localization and mapping, and motion planning and control [36].

Notably, the motion planning problem is ubiquitous in most of autonomous robotics applications, beside UAS. In simple words, motion planning is defined as the computation of the control input needed to drive a vehicle from an initial state to a target state satisfying the vehicle kinematic and dynamic constraints, while avoiding obstacles and other forbidden zones [21]. Typically, the motion planning problem is subdivided into two sub-problems: path planning and path tracking. Much effort has been devoted to deal with both tasks. The authors in [14] adopted a potential field approach for planning, followed by a multi-constrained Model Predictive Control (MPC) strategy for tracking. The authors in [7] proposed a two-stage approach where path planning is computed by leveraging the Rapidly-exploring Random Tree (RRT) algorithm, associated with a Linear Quadratic Regulator (LQR) controller for the tracking of the resulting reference trajectory. Similarly, in [26] the planned reference trajectory provided by the RRT algorithm was post-optimized using MPC to determine a feasible trajectory plan. However, none of the mentioned two-stage approaches guarantee the dynamic feasibility of the computed path.

Alternatively, another classical approach for path planning breaks the problem into two phases: a continuous

✉ Alessandro Rizzo
alessandro.rizzo@polito.it

[1] Department of Mechanical and Aerospace Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

[2] Department of Electronics and Telecommunications, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

Published online: 09 November 2021

collision-free path is generated in the first phase, while in the second phase a low-cost trajectory is computed along the previously generated path, respecting dynamic constraints. However, this approach may lead to infeasible or inefficient trajectories, due to the possible incompatibility of the cost function used by the optimization engine in the first and the second phase. For example, minimizing the Euclidean distance in the first phase may result in a continuous collision-free path that is incompatible with the dynamic constraints of the second phase. This becomes more evident when the motion planning is required to avoid obstacles in a time-varying environment. Hence, a more suitable approach would be to consider both the kinematic and dynamic constraints simultaneously, while constructing the trajectory during the planning phase. To address this issue, kinodynamic motion planning algorithms can be employed in order to satisfy at the same time both the kinematic and the dynamic constraints of the motion planning model [8]. A considerable number of kinodynamic strategies makes use of various primitive curves to define a path, such as Bezier curves [22], harmonic potential fields [27], or using learning approaches [25].

Recently, incremental sampling-based planners have been successfully employed to solve the kinodynamic motion planning problem as a two-point boundary value problem in the dynamic state space of the robot system. Sampling-based planners, such as Rapidly-exploring Random Tree (RRT) and the Probabilistic Roadmap (PRM), are employed even in high dimensional spaces to solve motion planning problems [23]. Generally, sampling-based planners search for the motion planning solution in one of the following spaces: the state space $\mathcal{X}$, representing the possible states of the vehicle; the control space $\mathcal{U}$, representing the possible control input configurations applied to drive the vehicle; and the reference space $\mathcal{Y}$, representing the possible desired vehicle states. A sampling-based kinodynamic algorithm was proposed for the first time by LaValle and Kuffner in [24], in which an RRT algorithm samples the control input of the vehicle and, then, predicts its corresponding motion in order to construct a tree of trajectories in the state space. As a consequence, the computed trajectory satisfies by design the constraints imposed by the vehicle dynamics and can be easily executed by the vehicle. Several sampling-based planners have been developed since then by extending and improving such approach [10, 13]. The authors in [17] proposed the RRT* algorithm, one of the most widely used sampling-based techniques. RRT* constitutes an improvement of the original RRT algorithm toward the attainment of a near optimal solution. Moreover, the RRT* algorithm has been further improved for specialized purposes such as real-time path planning [31], anytime planning [18], multi-agent planning [6], and others [32].

RRT* has improved the path quality of the original RRT through employing two new major mechanisms: rewiring and best neighbor search. RRT* incrementally adds new connections to the existing tree whenever a new sample is generated. Such a rewiring procedure gives RRT* the chance to gradually improve its path-cost, asymptotically approaching the lowest-cost path as the number of iterations increases. However, this rewiring is performed on a local basis, preventing a global propagation of the changes in the graph and a consequent optimization at the global level. An improvement is constituted by RRT#, which generates a guaranteed asymptotically optimal graph that always contains the lowest-cost path [2].

Another common concern in sampling-based approaches is related to the random sampling of the control space, instead of the random sampling of the reference space of the robot [20]. While sampling in the reference space always generates feasible trajectories, sampling in the control space may often result in the selection of inputs that can lead to infeasible trajectories, due to the presence of dynamic constraints. This typically yields longer execution times and an inefficient management of the algorithm. To address this problem, the authors in [20] proposed the *closed-loop* RRT (CL-RRT), in which the samples are drawn from the reference space instead of the control space. The sampled reference is then used to compute a trajectory using the closed-loop model of the robot. A similar approach is also used in [1] with the RRT# algorithm. Similarly, in [16] a kinodynamic RRT* is realized using Dubins curves as a primitive curve defining the reference path between the new sample and the existing tree, whereas an LQR controller is used in [33] to compute the cost of tracking the reference path.

This paper presents a kinodynamic motion planning algorithm called MP-RRT#, where RRT# [2] is enhanced by the use of Model Predictive Control (MPC) [5] to compute the cost between vertices evaluated in the rewiring of the new sample. The proposed sampling-based planner avoids the local rewiring limitation of RRT* by employing RRT# [2], and draws its samples from the reference space instead of the control space, as proposed in [20]. Then, given an input sample of the closed-loop system, i.e., a reference r in the reference space $\mathcal{Y}$, the proposed algorithm uses MPC [5] to perform a forward simulation obtaining a state trajectory and the optimal control input to track the sampled reference.

The proposed MP-RRT# concurrently constructs two graphs: *(i)* a graph $\mathcal{G}^{\mathcal{Y}} \in \mathcal{Y}$ to explore the reference space $\mathcal{Y}$, and *(ii)* a graph $\mathcal{G}^{\mathcal{X}} \in \mathcal{X}$ in the state space $\mathcal{X}$ to evaluate the UAS motion between vertices of $\mathcal{G}^{\mathcal{Y}}$ using the MPC strategy. Practically, $\mathcal{G}^{\mathcal{X}}$ consists of a graph of trajectories in the state space of the UAS.

The proposed strategy is more efficient than other kinodynamic RRT-based approaches [10, 13, 24], since it guarantees the feasibility of the trajectory for each sampled reference state passed to the MPC strategy. Obstacle avoidance is encapsulated in the approach by labeling trajectories containing obstacles as unfeasible and consequently discarding them, rather than frequently invoking a costly trajectory repairing due to the presence of obstacles. Hence, the proposed strategy requires fewer samples to construct an exploration tree and, consequently, less effort to compute an optimal trajectory. Additionally, the use of the MPC strategy guarantees that the computed trajectory satisfies input constraints. Anyway, the feasibility of the resulting trajectory is further examined later by the algorithm in a verification step to ensure collision avoidance. Importantly, sampling in the reference space $\mathcal{Y}$ rather than the state space $\mathcal{X}$ is generally more efficient, especially for vehicles with complex dynamics where the reference space has smaller dimension than the state space. For instance, here the UAS state space had a dimension of 8 variables, whereas the reference space had a dimension of 3 variables only.

Figure 1 illustrates the proposed strategy to generate a graph of feasible trajectories using the MP-RRT$^{\#}$ algorithm. Given a sample drawn from the reference space, MP-RRT$^{\#}$ incrementally extends the graph $\mathcal{G}^{\mathcal{Y}} \in \mathcal{Y}$ by adding a new vertex and an edge. Then, following a primitive curve (i.e., a Dubins curve in this work) the reference path connecting the new vertex to the existing graph is defined, and MPC is used to construct the corresponding trajectory in the graph $\mathcal{G}^{\mathcal{X}} \in \mathcal{X}$, respecting the vehicle dynamics and control input constraints. Note that obstacle avoidance is implemented for the computed trajectory rather than the edges, i.e., even if the edge crosses an obstacle, it will not be discarded unless its corresponding trajectory enters the obstacle space.

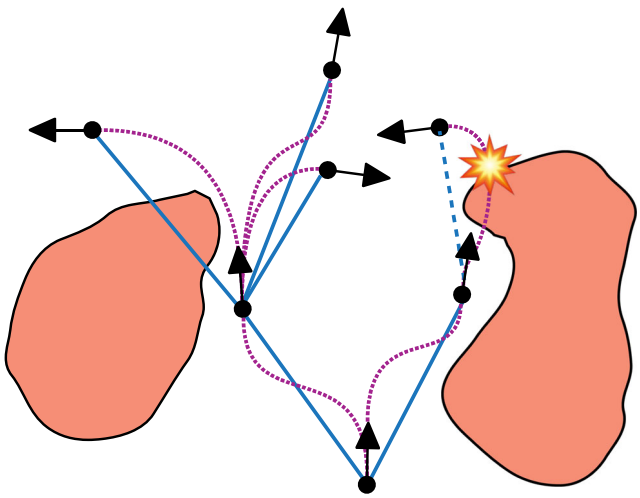


Fig. 1 Example of graphs constructed with MP-RRT$^{\#}$. The graph $\mathcal{G}^{\mathcal{Y}}$ consists of vertices (in black) and edges (in blue) in the reference state. On the other hand, the graph $\mathcal{G}^{\mathcal{X}}$ consists of trajectories (in magenta) obtained through evaluating the edges of $\mathcal{G}^{\mathcal{Y}}$ using the MPC strategy. An edge of $\mathcal{G}^{\mathcal{Y}}$ is labeled as invalid if its corresponding trajectory in $\mathcal{G}^{\mathcal{X}}$ crosses an obstacle

Here, the proposed MP-RRT$^{\#}$ strategy is specifically used to solve the motion planning problem of a multicopter to find a feasible trajectory for the UAS satisfying dynamic constraints. The novelty of the presented work resides in the use of the MPC in the graph construction of a RRT-based algorithm, which introduces evident benefits due to its ability of generating a forward path that is compatible with the vehicle constraints. In this way, the execution of feasible trajectories with maneuvers that are compatible with the vehicle constraints is guaranteed.

The rest of the paper is organized as follows. Section 2 defines the optimal motion planning problem. The proposed algorithm is described in Section 3, with a detailed description of the pseudocode, the UAS model and the MPC strategy adopted. Section 4 describes the experimental results, while our conclusions are drawn in Section 5.

2 Problem Formulation

This section defines the motion planning problem studied in this paper. First, the UAS dynamic model is described as

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

where $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ is the system state with dimension n_x, and $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ is the control input with dimension n_u. Both states and control inputs should respect specific constraints. Specifically, the vehicle state must belong to the *free state space* $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$, in order to navigate through an obstacle-free trajectory. This constraint is expressed by

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

where $\mathcal{X}$ is the state space and $\mathcal{X}_{\text{obs}}$ is the space occupied by obstacles. Moreover, the control input is constrained as

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

where $\mathcal{U}$ is the space of the admissible inputs (roll, pitch and thrust commands) in order to meet the mission specifications.

Given the initial state of the UAS $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ at time $t = 0$ and the target state defined by the goal region $\mathcal{X}_{\text{goal}} \subset \mathbb{R}^{n_x}$, the aim of the motion planning problem is to compute an optimal state trajectory $\overline{\boldsymbol{x}}^* : [0, t_f] \in \mathcal{X}_{\text{free}}$ and an optimal control input sequence $\overline{\boldsymbol{u}}^* : [0, t_f] \in \mathcal{U}$ over a finite time horizon from 0 to t_f able to drive the vehicle from the initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ to a final state within the goal region $\boldsymbol{x}(t_f) \in \mathcal{X}_{\text{goal}}$. $\overline{\boldsymbol{x}}^*$ and $\overline{\boldsymbol{u}}^*$ are computed minimizing a cost function Cost(·), while satisfying the constraints imposed

by Eqs. 2 and 3. Hence, the optimal motion is the solution of the following problem

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

Here, we will solve such a motion planning problem using the proposed MP-RRT$^{\#}$ algorithm. In particular, the proposed algorithm solves the motion planning problem in Eq. 4 by constructing a graph of optimal trajectories $\mathcal{G}^{\mathcal{X}}$ in the state space and, then, by selecting the best complete trajectory in the graph from the start to the goal state. Specifically, optimal trajectories of the graph $\mathcal{G}^{\mathcal{X}}$ are computed using a Model Predictive Control strategy that takes into account the UAS dynamic model in Eq. 1, satisfying the constraints imposed by Eqs. 2 and 3.

3 The MP-RRT$^{\#}$ Strategy

This section introduces the proposed MP-RRT$^{\#}$ algorithm (Model Predictive Rapidly-exploring Random Tree "sharp"), which enhances the RRT$^{\#}$ algorithm [2] using a MPC strategy to compute a near-optimal trajectory for UAS respecting the dynamic and kinematic constraints, while avoiding obstacles.

Similar to other kinodynamic RRT-based algorithms, our MP-RRT$^{\#}$ algorithm explores the search space by constructing an incremental graph rooted from the start. Specifically, the MP-RRT$^{\#}$ generates two graphs simultaneously: *(i)* $\mathcal{G}^{\mathcal{Y}}$ in the reference space $\mathcal{Y}$, and *(ii)* $\mathcal{G}^{\mathcal{X}}$ in the state space $\mathcal{X}$.

The former graph, i.e., $\mathcal{G}^{\mathcal{Y}}$, consists of vertices and edges in the reference space $\mathcal{Y}$. It is constructed incrementally by sampling vertices and growing the graph to uniformly explore the reference space. This is equivalent to any other graph generated to other algorithms of the RRT family. On the other hand, the latter graph, $\mathcal{G}^{\mathcal{X}}$, consists of a graph of trajectories computed through MPC. $\mathcal{G}^{\mathcal{X}}$ is built concurrently with $\mathcal{G}^{\mathcal{Y}}$ and, practically, it is used to evaluate the motion between vertices of $\mathcal{G}^{\mathcal{Y}}$, generating a graph of feasible trajectories in the state space. Figure 1 shows a simple example of the proposed strategy, where the graph $\mathcal{G}^{\mathcal{Y}}$ in blue is constructed in the reference state, while the corresponding graph $\mathcal{G}^{\mathcal{X}}$ in the state space is colored in magenta.

As can be observed in Fig. 1, collisions with obstacles are accounted for by graph $\mathcal{G}^{\mathcal{X}}$. If a trajectory in $\mathcal{G}^{\mathcal{X}}$ enters the obstacle space, the corresponding edge in $\mathcal{G}^{\mathcal{Y}}$ will not be included in the resulting graph. On the contrary, as can be observed in Fig. 1, even if an edge in $\mathcal{G}^{\mathcal{Y}}$ crosses an obstacle, it will not be discarded if its corresponding trajectory in $\mathcal{G}^{\mathcal{X}}$ does not collide with obstacles. This choice is motivated by the fact that, in general, the reference state has generally a lower dimension than the dimension of the vehicle state. As a consequence, it is more efficient to generate a graph in the reference space than in the state space.

3.1 Algorithm

The proposed algorithm is based on the the RRT$^{\#}$ algorithm proposed in [2]. The RRT$^{\#}$ is a special 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 $\mathcal{X}_{\text{goal}}$, the reference space $\mathcal{Y}$ and the state space $\mathcal{X}$ in which the motion planning searches for a feasible solution.

First, both graphs $\mathcal{G}^{\mathcal{Y}}$ and $\mathcal{G}^{\mathcal{X}}$ are initialized (lines 2 to 4). In particular, the initial vertex in the reference state is defined using the initial state x_0 (line 3). In fact, we assume that the reference space $\mathcal{Y}$ is a subset of the state space $\mathcal{X}$ and, as a consequence, an element $r \in \mathcal{Y}$ can be derived from a state $x \in \mathcal{X}$. 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 both graphs $\mathcal{G}^{\mathcal{X}}$ and $\mathcal{G}^{\mathcal{Y}}$ are extended by adding the new vertex (line 7). The Replan() function propagates this update on the graphs (line 8). Both the Extend() and Replan() functions are detailed in Algorithms 2 and 4, respectively. Finally, the branch $\mathcal{T}^{\mathcal{X}}$ connecting the initial and the target states is extracted from the graph $\mathcal{G}^{\mathcal{X}}$ (line 9) and returned as the solution of the algorithm.

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

```
1  MP-RRT# (x_0, X_goal, Y, X)
2      G^X ← {x_0};
3      r_0 ← x_0;
4      G^Y ← {r_0};
5      for i = 0 to N do
6          r_rand ← Sample();
7          G^X, G^Y ← Extend(G^X, G^Y, r_rand);
8          Replan(G^X, G^Y);
9      T^X ← SpanningTree(G^X);
10     return T^X
```

The Extend procedure is a crucial element for the proposed approach; it is responsible for the expansion of both graphs by adding a new vertex, after which the cost of the state trajectory is computed using MPC. This procedure

is detailed in Algorithm 2. Initially, the new vertex r is connected to the nearest vertex r_{nearest} in the graph $\mathcal{G}^{\mathcal{Y}}$ (line 2). Then, the Nearest() function finds the vertex with the minimum Euclidean distance from r. Hence, the states x_{nearest} and x are defined from r_{nearest} and r, respectively (lines 3 and 4). The ComputeTrajectory() function (line 5) uses MPC to compute the optimal state trajectory $\bar{\text{x}}$ moving from x_{nearest} to x. 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 adding the cost at the previous vertex to the cost of the trajectory $\bar{x}$, denoted by $c(\bar{x})$ (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 8 to 10).

The Near() function selects the M-nearest vertices as defined in [17]. Specifically, the number M of neighbors evaluated is defined as

$$M = e(1 + 1/d)\log|V|, \tag{5}$$

where d is the dimension of the reference space $\mathcal{Y}$, and the notation $|V|$ defines the cardinality of the set of vertices, i.e. the number of vertices in the graph $\mathcal{G}^{\mathcal{Y}}$. According to [17], Eq. 5 ensures the asymptotic optimality of the algorithm.

The FindParent() function searches for the neighbor vertex of r that provides the minimum *cost-to-come* $g()$ including the vertex r to the graph $\mathcal{G}^{\mathcal{Y}}$ and, similarly, the corresponding state x to the graph $\mathcal{G}^{\mathcal{X}}$ (line 11). Then, the vertex r is included in the priority queue q (line 12) used in the Replan() to propagate any updated cost in the graph $\mathcal{G}^{\mathcal{Y}}$.

The FindParent() procedure is detailed in Algorithm 3. For each near vertex of r, the state trajectory from x and x_{near} is computed to select the best parent vertex (from lines 2 to 9). Then, the selected r_{near} is defined as parent of r (line 8) and, similarly, x_{near} is defined as parent of x (line 9).

The priority queue has a crucial role in the RRT$^{\#}$ algorithm [2], since it is a queue of vertices that is evaluated in the 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 defined in the A* algorithm [12]

$$f(r) = g(r) + \hat{h}(r). \tag{6}$$

Function $g(r)$ represents the *cost-to-come* at the vertex r, i.e. the cost of moving between the start vertex r_0 and r, with $g(r_0) = 0$. Function $\hat{h}(r)$ is the estimated *cost-to-go* to reach the goal state, with $\hat{h}(r_{\text{goal}}) = 0$.

In particular, the Replan() procedure is detailed in Algorithm 4. This procedure is based on an iterative loop that updates only promising vertices (lines 2 to 15), i.e., vertices that can improve the current solution in the graph.

Algorithm 2 The Extend procedure.

```
1  Extend(G^X, G^Y, r)
2      r_nearest ← Nearest(G^Y, r);
3      x_nearest ← r_nearest;
4      x ← r;
5      x̄ ← ComputeTrajectory(x_nearest, x);
6      if isTrajectoryValid(x̄) then
7          g(r) ← g(r_nearest) + c(x̄);
8      N(r) ← Near(G^Y, r);
9      foreach r_near ∈ N(r) do
10         N(r_near) ← N(r_near) ∪ {r};
11     FindParent(r, x);
12     UpdateQueue(r);
13     return G^X, G^Y
```

Specifically, the set of promising vertices $V_{\text{prom}} \subset V$ contains vertices inside the relevant region $\mathcal{Y}_{\text{rel}} \in \mathcal{Y}$

$$\mathcal{Y}_{\text{rel}} = \{r \in \mathcal{Y}_{\text{free}} : f(r) < g(r^*_{\text{goal}})\}, \tag{7}$$

where r^*_{goal} is the vertex in the goal region with the minimum *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 cannot improve the current solution, speeding up the algorithm. The first element of the queue is selected (line 3) 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 15) as a new parent vertex. This is verified by computing the *cost-to-come* of the resulting state trajectory of moving from x to x_{nbh}. Similar to Algorithm 3, line 10 checks if the neighbor vertex $r_{\text{nbh}} \in N$ is a promising vertex and, in line 11, if r can be the new parent vertex of r_{nbh}. If this condition occurs, the vertex r_{nbh} is included in q to be evaluated in the Replan() procedure. In

Algorithm 3 The FindParent procedure.

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

particular, r_{nbh} is defined in the reference space, while x_{nbh} is the corresponding state in the state space.

Algorithm 4 The Replan procedure.

```
Replan(G^X, G^Y)
  while f(q.top()) ≺ g(r*_goal) do
    r = q.top();
    x ← r;
    q.pop();
    foreach r_nbh ∈ N(r) do
      x_nbh ← r_nbh;
      x̄ ← ComputeTrajectory(x, x_nbh);
      if isTrajectoryValid(x̄) then
        if g(r) + c(x̄) + ĥ(r_nbh) < g(r*_goal) then
          if g(r) + c(x̄) < g(r_nbh) then
            g(r_nbh) = g(r) + r(x̄);
            P(r_nbh) = r;
            P(x_nbh) = x;
            UpdateQueue(r_nbh);
```

3.2 UAS Model

Here, we adopt a multicopter, with a dynamic motion model linearized around its hovering condition [15], in which small variations of the attitude angle are assumed and the vehicle heading is aligned with the x-axis of the multicopter inertial frame. Hence, the linear model of the UAS is defined as

$$\dot{x}(t) = A_c x(t) + B_c u(t), \tag{8}$$

where A_c is the state matrix in continuous time

$$A_c = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -a_x & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & -a_y & 0 & 0 & -g \\ 0 & 0 & 0 & 0 & 0 & -a_z & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} \end{bmatrix}, \tag{9}$$

B_c is the input matrix in continuous time

$$B_c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{k_\phi}{\tau_\phi} & 0 & 0 \\ 0 & \frac{k_\theta}{\tau_\theta} & 0 \end{bmatrix}, \tag{10}$$

and 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 commands in the inertial frame and T is the thrust control command.

Since we will use a controller implemented in the discrete time, the UAS model is discretized as

$$A = e^{A_c T_s}, \tag{11}$$

$$B = \int_0^{T_s} e^{A_c d\tau} d\tau B_c, \tag{12}$$

where T_s is the sampling time.

3.3 Model Predictive Control

In order to construct the graph $\mathcal{G}^{\mathcal{X}} \in \mathcal{X}$ incrementally, the MP-RRT$^{\#}$ algorithm uses MPC to compute the optimal state trajectory between every newly added vertex and its adjacent vertices and, then, to evaluate the cost of such a trajectory.

Based on the UAS model previously defined, in this work we implement a Linear Model Predictive Control inspired by [15].

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

$$\begin{aligned} J(\overline{x}, \overline{u}) = \Bigg(& \sum_{\text{k}=0}^{H_\text{p}-1} (x_\text{k} - x_{\text{ref,k}})^T Q_\text{x} (x_\text{k} - x_{\text{ref,k}}) \\ & + (u_\text{k} - u_{\text{k}-1})^T R_\Delta (u_\text{k} - u_{\text{k}-1}) \Bigg) \\ & + (x_{H_\text{p}} - x_{\text{ref},H_\text{p}})^T Q_{\text{final}} (x_{H_\text{p}} - x_{\text{ref},H_\text{p}}), \end{aligned} \tag{13}$$

where H_p is the prediction horizon. The input vector is $\overline{u} = [u_0\, u_1 \ldots u_{H_\text{p}}]^T$, with $u_\text{k} \in \mathbb{R}^3$, for $k = 0, \ldots, H_\text{p} - 1$. The state vector is $\overline{x} = [x_0\ x_1\ \ldots\ x_{H_\text{p}}]^T$, with $x_\text{k} \in \mathbb{R}^8$, for $k = 0, \ldots, H_\text{p}$. The vector reference state is $\overline{x}_\text{ref} = \begin{bmatrix} x_{\text{ref},0}\ x_{\text{ref},1}\ \ldots\ x_{\text{ref},H_\text{p}} \end{bmatrix}^T$, with $x_{\text{ref,k}} \in \mathbb{R}^8$, for $k = 0, \ldots, H_\text{p}$. Matrices Q_x, R_Δ and Q_final are positive semidefinite matrices indicating the penalty matrix on the state error, the penalty matrix on the variation of the control input, and the terminal cost matrix on the last state error. The computation of Q_final is carried out by iteratively solving a suitable Algebraic Riccati Equation [4].

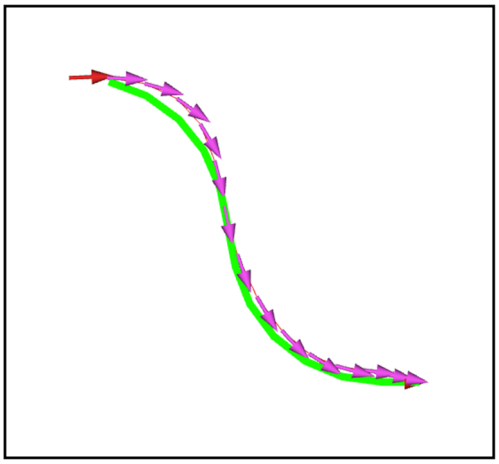


Fig. 2 Example of reference trajectory computed using Dubins curves and connecting two adjacent vertices. The green line is the reference trajectory, whereas magenta arrows are the state trajectory computed using MPC

Hence, the following convex optimization problem is solved

$$\overline{x}^*, \overline{u}^* = \min_{U,X} \quad J(\overline{x}, \overline{u}) \tag{14}$$

$$\text{subject to} \quad x_{k+1} = Ax_k + Bu_k \tag{15}$$

$$u_k \in \mathcal{U} \tag{16}$$

$$x_0 = x(t_0) \tag{17}$$

The optimization problem of Eq. 14 requires a reference trajectory x_{ref}. In this work, the reference trajectory is defined using Dubins curves [9]. Compared to other primitive curves used to find the shortest path between two configurations, and for our purposes of building the graph iteratively from curve segments extending the existing path with a new vertex, Dubins curves are a suitable solution to achieve flyable paths, since they are forward-only curves, whereas other curves like Reeds-Shepp require backward-motion [23]. The Dubins curvature radius generally affects motion planning, whereby different curvature radii yield different planned paths. As a fundamental requisite, the curvature radius should reflect the minimum curvature that the vehicle can execute, compatible with kinodynamic constraints. Here, we are considering a multicopter that, theoretically, has zero curvature radius (i.e., it can rotate around its axis in place). However, since we are assuming a constant nonzero cruise velocity in the motion planning, a lower bound for the curvature radius needs to be set.

Dubins curves refer to the shortest path between two poses in the two-dimensional space considering a constant radius curvature. This solution fits perfectly with our work, since the algorithm is implemented in the two-dimensional space. However, Dubins curves are also extended to the three-dimensional space [29] and with a variable radius curvature [11] being able to be used also in more complex scenarios.

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 equations of Dubins curves are

$$\dot{p}_x = \cos(p_\beta), \tag{18}$$

$$\dot{p}_y = \sin(p_\beta), \tag{19}$$

$$\dot{p}_\beta = u_c, \tag{20}$$

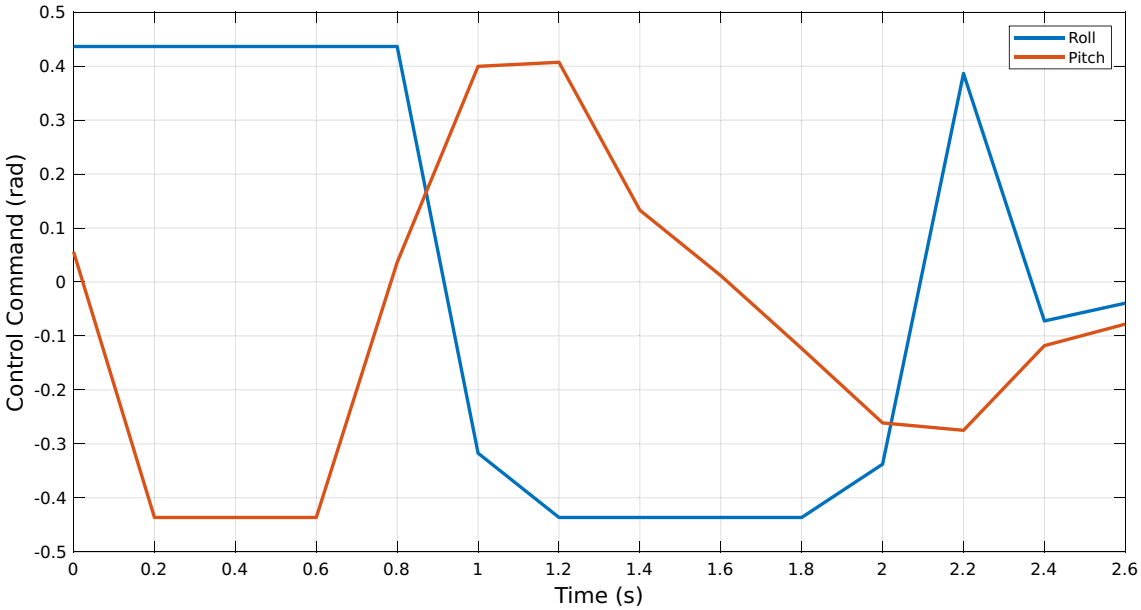


Fig. 3 The roll and pitch control inputs computed by MPC to follow the trajectory of Fig. 2

Table 1 Parameters used for the UAS model

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

where u_c is normalized in the range between -1 and 1 with respect to the maximum curvature radius of the aircraft. The shortest path between two poses can be expressed as a combination of no more than three motion primitives [9]. 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), $u_c = -1$ describes a right (R) turn, and $u_c = 1$ describes a left (L) turn, thus obtaining six possible curves

$$\{LRL, RLR, LSL, LSR, RSL, RSR\}. \quad (21)$$

The optimization problem of Eq. 14 is solved following the reference trajectory. Then, in accordance with the MPC philosophy, only the first control input is applied and the optimization is solved iteratively. Figure 2 shows an example of reference trajectory generated using Dubins curves and followed through MPC. Figure 3 illustrates the roll and pitch control commands computed to follow the trajectory.

4 Results

4.1 Implementation

The proposed strategy is implemented in C++ using the Robot Operating System (ROS) [34] framework and the Open Motion Planning Library (OMPL) [38], which provides many state-of-the-art sampling-based algorithms and many additional functionalities to facilitate the development of new algorithms.

The MP-RRT$^\#$ algorithm is implemented considering a two-dimensional space, i.e., flying at a fixed altitude. Hence, the selected configuration space is the Special

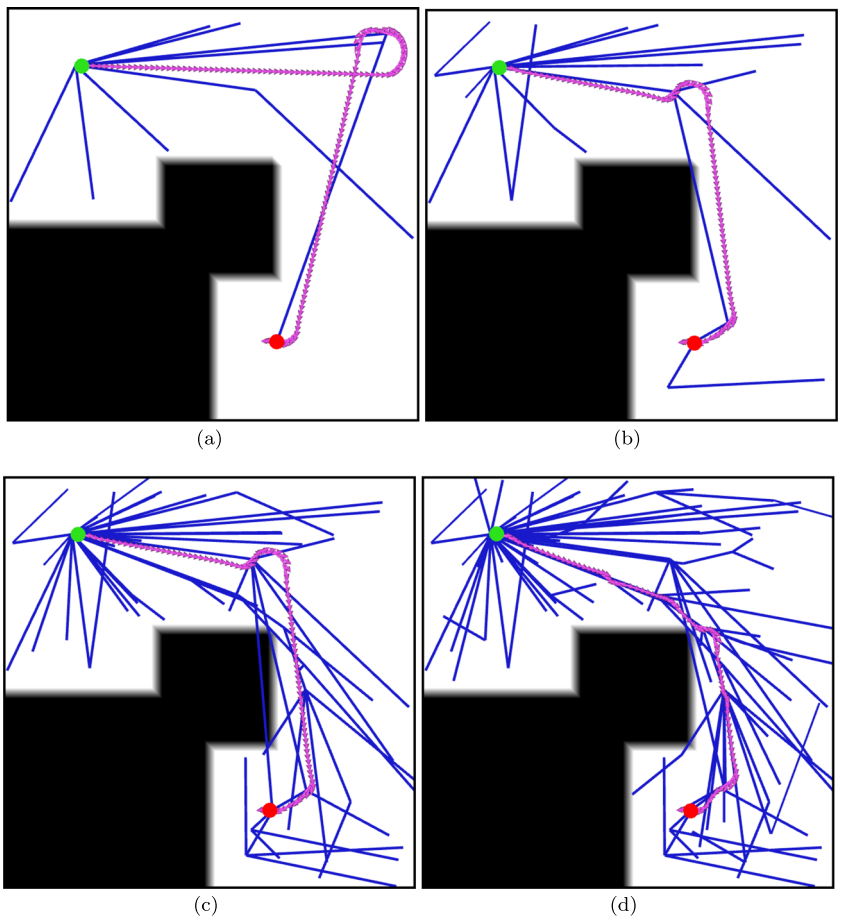


Fig. 4 The construction of the exploration tree using the MP-RRT$^\#$ algorithm. The start and target positions are in green and in red, respectively. The graph $\mathcal{G}^{\mathcal{Y}}$ in the reference space is colored in blue, while the computed path obtained from the graph $\mathcal{G}^{\mathcal{X}}$ in the state space is colored in magenta. In (a), the graph consists of 10 vertices rooted from the start pose finding an initial solution in the map with a cost (i.e., the path length) of 66.44 m. In (b), the graph with 20 vertices, in which the solution is improved with a cost of 45.09 m. In (c), the graph consists of 60 vertices, but the solution is not improved. In (d), the graph has 100 vertices obtaining a solution with cost 38.70 m

Euclidean Group $SE(2)$ in which each admissible configuration is a pose in the two-dimensional space free to translate and rotate. Each reference sampled by the algorithm in the reference space $\mathcal{Y}$ consists therefore of three parameters, i.e., two defining the position of the UAS and one defining its orientation, corresponding to the flight direction. Each 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 track it.

The motion-cost of the trajectory is computed considering the path length of the resulting trajectory

$$c(\overline{x}, \overline{u}) = \sum_{i=1}^{M} \|x_{\mathrm{i}} - x_{\mathrm{i}-1}\|_2, \tag{22}$$

with $x_{\mathrm{i}} \in \overline{x}$, and M is the size of the trajectory. On the other hand, the *cost-to-go* $\hat{h}(r)$ is computed as the length of the Dubins curve between the vertex r and the goal region $\mathcal{X}_{\mathrm{ref}}$.

The optimization problem of the MPC is solved using CVXGEN [28], 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 [28]. Hence, the linear model of the UAS and the Linear MPC problem of Eqs. 13 and 14 are included and solved with CVXGEN.

Experimental tests are performed considering the multicopter Asctec Firefly and using the parameters listed in Table 1.

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

$$-0.436 \text{ rad} \leq {}^{\mathbb{W}}\phi_d \leq 0.436 \text{ rad} \tag{23}$$

$$-0.436 \text{ rad} \leq {}^{\mathbb{W}}\theta_d \leq 0.436 \text{ rad} \tag{24}$$

$$-4.80 \text{ N} \leq T \leq 10.19 \text{ N}. \tag{25}$$

The MPC is manually tuned by setting matrices Q_{x} and R_{Δ} through trial-and-error to attain a satisfactory behavior in tracking the reference trajectory. Specifically, Q_{x}, and R_{Δ} are set as

$$Q_{\mathrm{x}} = \begin{bmatrix} 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 40 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{26}$$

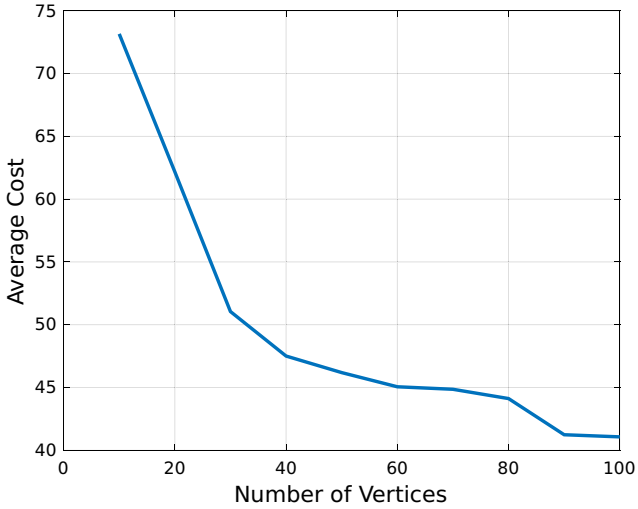


Fig. 5 The average cost of the solution path against the number of vertices in the MP-RRT$^{\#}$ algorithm. The average cost is computed running the algorithm 50 times in the same scenario of Fig. 4

$$R_{\Delta} = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.0025 \end{bmatrix}. \tag{27}$$

Table 2 Trajectory tracking performance indices collected over 20 trajectories

Trajectory	Length [m]	Vertices	Avg Tracking Error [m]
1	47.178677	17	0.04951
2	43.488594	17	0.04924
3	49.000358	16	0.04934
4	48.921484	16	0.04940
5	48.720384	18	0.04930
6	47.986777	19	0.05003
7	46.894054	16	0.04931
8	44.604293	15	0.04949
9	51.149531	16	0.04875
10	45.98843	15	0.04938
11	48.523593	18	0.04869
12	47.661833	15	0.05003
13	43.958282	16	0.04825
14	43.825664	14	0.04939
15	49.610042	16	0.04799
16	48.843731	15	0.04838
17	49.636594	17	0.04898
18	45.822011	15	0.04950
19	45.370298	16	0.04885
20	45.527057	16	0.05007

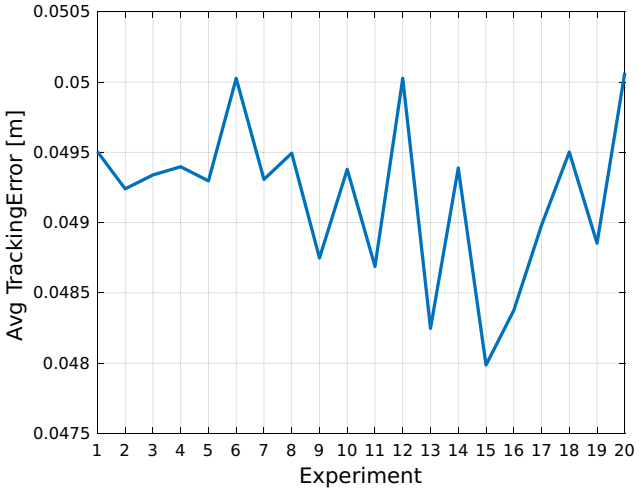


Fig. 6 The average tracking error for 20 trajectories running the same scenario of Fig. 4

Moreover, Q_{final} is computed by iteratively solving the Algebraic Riccati Equation [4].

Figure 2 illustrates an example of reference trajectory computed with Dubins curves and connecting two vertices. The trajectory is followed by the MPC that reaches the target vertex computing the roll and pitch control commands plotted in Fig. 3.

4.2 Simulation Results

The proposed MP-RRT$^{\#}$ algorithm is tested in different scenarios to evaluate its behavior in computing UAS trajectories.

Figure 4 shows the evolution of the graph during the exploration of the reference space (i.e., the map). Specifically, in Fig. 4(a), the algorithm computes a graph with 10 vertices finding an initial solution that is far from the optimal one. In Fig. 4(b), the graph consists of 20 vertices, improving the solution path. On the contrary, the solution is not improved in Fig. 4(c), with a graph with 60 vertices. Finally, in Fig. 4(d), a better solution is found with a graph with 100 vertices. The previously described test highlights the ability of the proposed algorithm to explore the map and to compute a feasible trajectory for the UAS. The quality of the computed trajectory increases with the number of vertices in the graph, converging toward the optimal solution. In order to demonstrate the above mentioned pattern, we performed 50 tests using the same scenario of Fig. 4. The average cost of the resulting solution path against the number of iterations of the MP-RRT$^{\#}$ algorithm is shown in Fig. 5.

Considering the same scenario of Fig. 4, we evaluate the ability of the MPC in tracking the reference trajectory

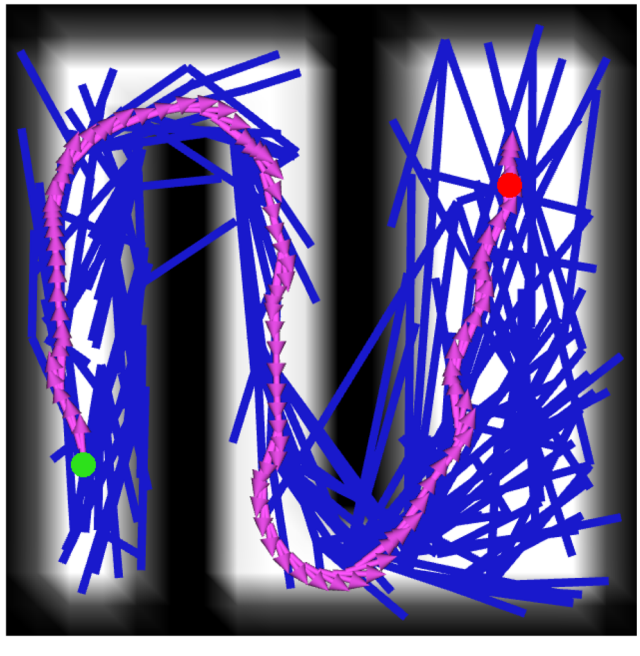
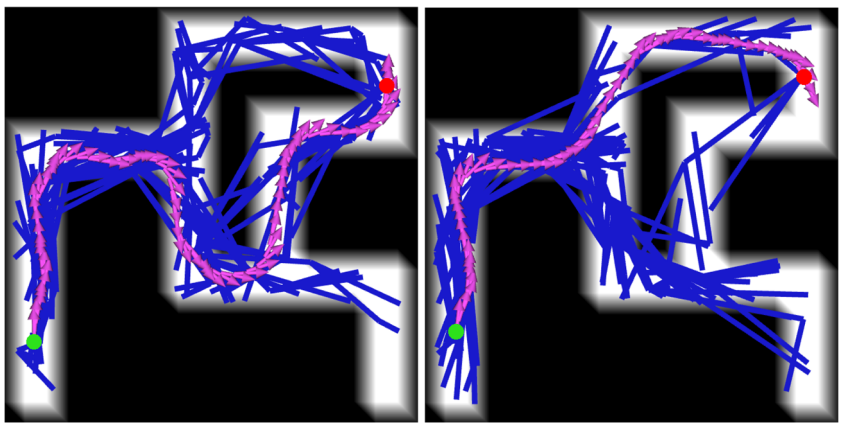


Fig. 8 Example of trajectory computed with the MP-RRT$^{\#}$ by constructing a graph with 400 vertices. The start and target positions are in green and in red, respectively. In blue the graph $\mathcal{G}^{\mathcal{Y}}$ in the reference space, while in magenta the computed path obtained from the graph $\mathcal{G}^{\mathcal{X}}$ in the state space

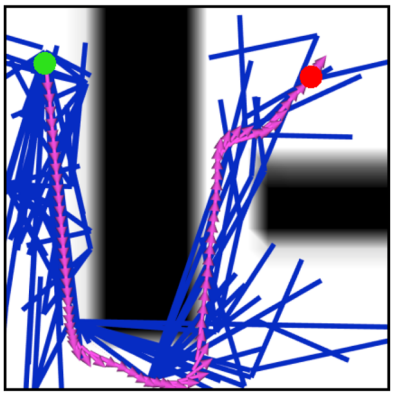


(a) (b)

Fig. 7 Trajectories computed with the MP-RRT$^{\#}$ by constructing a graph with 400 vertices. The start and target positions are in green and in red, respectively. In blue, the graph $\mathcal{G}^{\mathcal{Y}}$ in the reference space, while in magenta, the computed path obtained from the graph $\mathcal{G}^{\mathcal{X}}$ in the state space. In (a) and (b), the target is in a similar position, but with opposite orientation. As a consequence, the solution is completely different, yielding different paths

defined using Dubins curves. Table 2 reports the average trajectory tracking error in 20 tests. The average tracking error is the average Euclidean distance between the setpoints of the reference trajectory defined using Dubins curves and their corresponding states in the actual state trajectory computed using the MPC and satisfying dynamic constraints. The average tracking error along the whole path was found to be reasonably small, being always smaller than 0.05 m along trajectories with a length ranging between 43 and 51 m. Figure 6 illustrates the average tracking error for each of the 20 tests.

Other tests in more complex maps are shown in Figs. 7 and 8. In particular, Fig. 7 shows an interesting scenario, in which Figs. 7(a) and (b) present the target in similar positions but with opposite directions. As a consequence, the algorithm computes different solutions in order to reach the target with the desired flight direction.



(a)

(b)

Fig. 9 In (a), the trajectory computed with the MP-RRT$^{\#}$ algorithm by constructing a graph of 100 vertices. In (b), the computed trajectory is executed by the PX4 autopilot in a simulation

Similarly, in Fig. 9(a), the MP-RRT$^{\#}$ algorithm explores a map with a graph of 100 vertices, computing a solution. The trajectory computed in Fig. 9(a) is also executed in a realistic simulation environment, using Gazebo and SITL frameworks. Gazebo is an open-source multi-robot simulator fully compatible with ROS [19] able to simulate robots, sensors, and rigid body dynamics. SITL (Software In The Loop) [37] 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 [30], 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 Fig. 9(b). Although the environment of Fig. 9(b) does not correspond to the map of Fig. 9(a), the executed trajectory in Fig. 9(b) is the same generated in Fig. 9(b).

5 Conclusions

In this paper, we have introduced a novel kinodynamic sampling-based motion planning algorithm called MP-RRT$^{\#}$, which enhances the RRT$^{\#}$ using a Model Predictive Control strategy to compute an optimal trajectory for UAS.

Similar to RRT$^{\#}$, the proposed algorithm explores the map constructing an asymptotically optimal graph. Specifically, two graphs are concurrently constructed: $\mathcal{G}^{\mathcal{Y}}$ and $\mathcal{G}^{\mathcal{X}}$. First, the graph $\mathcal{G}^{\mathcal{Y}}$ explores the reference space of the UAS. Then, the MPC strategy is used to iteratively evaluate the feasibility of each newly added vertex and to compute the cost of its corresponding edge constructing a graph $\mathcal{G}^{\mathcal{X}}$ of feasible trajectories in the state space. The resulting trajectory computed by the proposed MP-RRT$^{\#}$ algorithm is a near-optimal trajectory that respects both the kinematic and dynamic constraints of the UAS.

The proposed MP-RRT$^{\#}$ algorithm differs from other kinodynamic RRT-based algorithms in sampling the input reference of the closed loop system instead of directly sampling the control input. This gives rise to considerable advantages, especially when dealing with vehicles with complex dynamics where the reference space dimension is considerably smaller than the control space and state space of the vehicle.

The simulation results obtained from the implementation of the proposed MP-RRT$^{\#}$ algorithm demonstrate good trajectory quality even for complex maps. Moreover, the computed trajectory is executable by a UAS equipped with a professional autopilot.

Although the proposed algorithm is tested in a simplified scenario, i.e., in a two-dimensional space using a linearized model of the UAS, the proposed MP-RRT$^{\#}$ algorithm

can be extended to more complex scenarios by increasing the complexity of the algorithm. Moreover, although the work presented here focuses on UAS, the proposed motion planning strategy has a general validity, and can be easily adapted to other kinds of robots, such as ground robots, autonomous cars, and underwater vehicles.

Future works will extend the current algorithm to solve a three-dimensional motion planning problem. The proposed MP-RRT$^{\#}$ strategy will be adapted for real-time motion planning problems like the one described in [1, 20]. In addition to that, experimental tests will be conducted on a physical robotic platform to evaluate the performance under realistic conditions.

Author Contributions SP conceived the research and designed a first version of the algorithm, AO collaborated to the development of the algorithm, implemented the Model Predictive Control Strategy, performed the simulations, and evaluated the results. SP and AO drafted a first version of the manuscript. AR supervised the research and produced a revised version of the manuscript. All the authors finally revised and agreed on the final version of the manuscript.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement. This work is partially supported by Compagnia di San Paolo and by an Amazon Research Award granted to Dr. A. Rizzo.

Data Availability Not applicable, as no real data have been used to realize this work.

Code Availability The code will be made available upon request.

Declarations

Ethics approval Not applicable. (This study does not involve human participants, their data or biological material).

Consent to participate and for publication The manuscript is approved by all authors for publication.

Conflict of Interests The authors declare neither conflict of interest, nor competing interests.

References

1. Arslan, O., Berntorp, K., Tsiotras, P.: Sampling-based algorithms for optimal motion planning using closed-loop prediction. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 4991–4996. IEEE (2017)
2. Arslan, O., Tsiotras, P.: Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: 2013 IEEE International Conference on Robotics and Automation (ICRA), pp. 2421–2428. IEEE (2013)
3. Bloise, N., Primatesta, S., Antonini, R., Fici, G.P., Gaspardone, M., Guglieri, G., Rizzo, A.: A survey of unmanned aircraft system technologies to enable safe operations in urban areas. In: 2019 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 433–442. IEEE (2019)
4. Borrelli, F., Bemporad, A., Morari, M.: Predictive Control for Linear and Hybrid Systems. Cambridge University Press, Cambridge (2017)
5. Camacho, E.F., Alba, C.B.: Model predictive control. Springer Science & Business Media (2013)
6. Čáp, M., Novák, P., Vokřínek, J., Pěchouček, M.: Multi-agent RRT*: Sampling-based cooperative pathfinding. arXiv:1302.2828 (2013)
7. Chen, Y., Peng, H., Grizzle, J.W.: Fast trajectory planning and robust trajectory tracking for pedestrian avoidance. IEEE Access **5**, 9304–9317 (2017)
8. Donald, B., Xavier, P., Canny, J., Reif, J.: Kinodynamic motion planning. J. ACM (JACM) **40**(5), 1048–1066 (1993)
9. Dubins, L.E.: On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. Am. J. Math. **79**(3), 497–516 (1957)
10. Frazzoli, E., Dahleh, M.A., Feron, E.: Real-time motion planning for agile autonomous vehicles. J Guid. Control Dyn. **25**(1), 116–129 (2002)
11. Hansen, K.D., la Cour-Harbo, A.: Waypoint planning with Dubins curves using genetic algorithms. In: 2016 European Control Conference (ECC), pp. 2240–2246. IEEE (2016)
12. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
13. Howard, T.M., Kelly, A.: Optimal rough terrain trajectory generation for wheeled mobile robots. Int. J. Robot. Res. **26**(2), 141–166 (2007)
14. Ji, J., Khajepour, A., Melek, W.W., Huang, Y.: Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints. IEEE Trans. Veh. Technol. **66**(2), 952–964 (2016)
15. Kamel, M., Burri, M., Siegwart, R.: Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles. IFAC-PapersOnLine **50**(1), 3463–3469 (2017)
16. Karaman, S., Frazzoli, E.: Optimal kinodynamic motion planning using incremental sampling-based methods. In: 49th IEEE Conference on Decision and Control (CDC), pp. 7681–7687. IEEE (2010)
17. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
18. Karaman, S., Walter, M.R., Perez, A., Frazzoli, E., Teller, S.: Anytime motion planning using the RRT. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 1478–1483. IEEE (2011)
19. Koenig, N.P., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 4, pp. 2149–2154. IEEE (2004)

20. Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., How, J.P.: Real-time motion planning with applications to autonomous urban driving. IEEE Trans. Control Syst. Technol. **17**(5), 1105–1118 (2009)
21. Latombe, J.C.: Robot motion planning, vol. 124. Springer Science & Business Media (2012)
22. Lau, B., Sprunk, C., Burgard, W.: Kinodynamic motion planning for mobile robots using splines. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2427–2433. IEEE (2009)
23. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
24. LaValle, S.M., Kuffner, J.r., J. J.: Randomized kinodynamic planning. Int. J. Robot. Res. **20**(5), 378–400 (2001)
25. Li, L., Miao, Y., Qureshi, A.H., Yip, M.C.: MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints. arXiv:2101.06798 (2021)
26. Lin, P., Chen, S., Liu, C.: Model predictive control-based trajectory planning for quadrotors with state and input constraints. In: 2016 16th International Conference on Control, Automation and Systems (ICCAS), pp. 1618–1623. IEEE (2016)
27. Masoud, A.A.: Kinodynamic motion planning. IEEE Robot. Autom. Mag. **17**(1), 85–99 (2010)
28. Mattingley, J., Boyd, S.: CVXGEN: A code generator for embedded convex optimization. Optim. Eng. **13**(1), 1–27 (2012)
29. McLain, T., Beard, R.W., Owen, M.: Implementing Dubins airplane paths on fixed-wing (UAVs) (2014)
30. Meier, L., Honegger, D., Pollefeys, M.: PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 6235–6240. IEEE (2015)
31. Naderi, K., Rajamäki, J., Hämäläinen, P.: RT-RRT* a real-time path planning algorithm based on RRT. In: Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games (MIG), pp. 113–118 (2015)
32. Noreen, I., Khan, A., Habib, Z., et al.: Optimal path planning using RRT* based approaches: a survey and future directions. Int. J. Adv. Comput. Sci. Appl **7**(11), 97–107 (2016)
33. Perez, A., Platt, R., Konidaris, G., Kaelbling, L., Lozano-Perez, T.: LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In: 2012 IEEE International Conference on Robotics and Automation (ICRA), pp. 2537–2542. IEEE (2012)
34. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3, p. 5 (2009)
35. Shakhatreh, H., Sawalmeh, A.H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., Othman, N.S., Khreishah, A., Guizani, M.: Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges. IEEE Access **7**, 48572–48634 (2019)
36. Siegwart, R., Nourbakhsh, I.R., Scaramuzza, D.: Introduction to autonomous mobile robots. MIT press (2011)
37. SITL contributors: SITL guide http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html (2020)
38. Şucan, I.A., Moll, M., Kavraki, L.E.: The open motion planning library. IEEE Robot. Autom. Mag. **19**(4), 72–82 (2012). https://doi.org/10.1109/MRA.2012.2205651. http://ompl.kavrakilab.org
39. Tang, S., Kumar, V.: Autonomous flight. Annual Review of Control, Robotics, and Autonomous Systems **1**, 29–52 (2018)

Stefano Primatesta is a postdoctoral Assistant Researcher in the Department of Mechanical and Aerospace Engineering. He received his Ph.D. in Computer and Control Engineering from Politecnico di Torino in 2019, his M.Sc. in Mechatronic Engineering, and his B.Sc. in Electronic Engineering from Politecnico di Torino in 2014 and 2011, respectively. His field of research is the use of Remotely Piloted Aircraft Systems in urban environments including virtual modeling and multi-dimensional risk analysis. His research interests include also autonomous navigation and service robotics, with applications to unmanned aerial vehicles and unmanned ground vehicles.

Abdalla Osman is a Ph.D. student in the Department of Electronics and Telecommunications at Politecnico di Torino, Italy. He received his M.Sc. in Mechatronic Engineering from Politecnico di Torino in 2016. In 2017, he started his Ph.D. within in the Complex Systems Laboratory, directed by Prof. Alessandro Rizzo, and the PIC4SeR (Interdipartimental center for service robotics), directed by Prof. Marcello Chiaberge, researching on computer vision and control systems for robotic applications.

Alessandro Rizzo received the Laurea degree (summa cum laude) in computer engineering and the Ph.D. degree in automation and electronics engineering from the University of Catania, Italy, in 1996 and 2000, respectively. In 1998, he worked as a EURATOM Research Fellow with JET Joint Undertaking, Abingdon, U.K., researching on sensor validation and fault diagnosis for nuclear fusion experiments. In 2000 and 2001, he worked as a Research Consultant at ST Microelectronics, Catania Site, Italy, and as an Industry Professor of robotics with the University of Messina, Italy. From 2002 to 2015, he was a tenured Assistant Professor with the Politecnico di Bari, Italy. Since 2012, he has been a Visiting Professor with the New York University Tandon School of Engineering, Brooklyn, NY, USA.

In November 2015, he joined Politecnico di Torino, where he is an Associate Professor in the Department of Electronics and Telecommunications and established the Complex Systems Laboratory. Dr. Rizzo is engaged in conducting and supervising research on complex networks and systems, modeling and control of nonlinear systems, and cooperative robotics. He is the author of two books, two international patents, and about 190 papers on international journals and conference proceedings. He has been a recipient of the Award for the Best Application Paper at the IFAC world triennial conference in 2002 and of the Award for the Most Read Papers in Mathematics and Computers in Simulation (Elsevier) in 2009. He has also been a Distinguished Lecturer of the IEEE Nuclear and Plasma Science Society and one of the recipients of the 2019 Amazon Research Awards.