NONLINEAR ENGINEERING AND ROBOTICS

MSC 2010: 12Y05, 49Kxx, 68W27

Path Planning Followed by Kinodynamic Smoothing for Multirotor Aerial Vehicles (MAVs)

G. Kulathunga, D. Devitt, R. Fedorenko, A. Klimchik

Any obstacle-free path planning algorithm, in general, gives a sequence of waypoints that connect start and goal positions by a sequence of straight lines, which does not ensure the smoothness and the dynamic feasibility to maneuver the MAV. Kinodynamic-based motion planning is one of the ways to impose dynamic feasibility in planning. However, kinodynamic motion planning is not an optimal solution due to high computational demands for real-time applications. Thus, we explore path planning followed by kinodynamic smoothing while ensuring the dynamic feasibility of MAV. The main difference in the proposed technique is not to use kinodynamic planning when finding a feasible path, but rather to apply kinodynamic smoothing along the obtained feasible path. We have chosen a geometric-based path planning algorithm “RRT*” as the path finding algorithm. In the proposed technique, we modified the original RRT* introducing an adaptive search space and a steering function that helps to increase the consistency of the planner. Moreover, we propose a multiple RRT* that generates a set of desired paths. The optimal path from the generated paths is selected based on a cost function. Afterwards, we apply kinodynamic smoothing that will result in a dynamically feasible as well as obstacle-free path. Thereafter, a b-spline-based trajectory is generated to maneuver the vehicle autonomously in unknown environments. Finally, we have tested the proposed technique in various simulated environments. According to the experiment results, we were able to speed up the path planning task by 1.3 times when using the proposed multiple RRT* over the original RRT*.

Keywords: RRT*, iLQR, B-spline, OctoMap, ellipsoidal search space

Received March 26, 2021
Accepted October 27, 2021

The work presented in this paper has been supported by Innopolis University, Ministry of Education and National Technological Initiative, within the framework of creation of the Center for Technologies in Robotics and Mechatronics Components.

Geesara Kulathunga
ggeesara@gmail.com
Dmitry Devitt
d.dmitry@innopolis.ru
Roman Fedorenko
r.fedorenko@innopolis.ru
Alexandr Klimchik
a.klimchik@innopolis.ru
Center for Technologies in Robotics and Mechatronics Components, Innopolis University
ul. Universitetskaya 1, Innopolis, 420500 Russia

1. Introduction

With the recent research advances in microcontroller technology and sensors' capabilities, a new era has begun for MAVs. In recent years, such devices have been applied in many fields including delivery, farming and cinematography. Since most of the listed scenarios have to perform real-time operations, optimal motion planning receives a lot of attention and plays a key role. Optimal motion planning consists of two parts: planning an obstacle-free path and further optimizing such a path, ensuring the provided constraints to obtain an optimal control policy to navigate MAV appropriately. Subsequently, geometric-based motion planning is a well-matured technique, although no differential constraints (i.e., vehicle dynamics) are considered. Conversely, kinodynamic motion planning is one of the ways to ensure vehicle dynamics feasibility and fulfill all the provided constraints. When considering real-time motion planning, geometric-based path planning followed by path parameterization is a well-adapted technique. However, in such an approach, path parameterization may fail despite selecting a geometrically shortest path.

In this paper, we explore kinodynamic smoothing followed by trajectory generation to keep the dynamic feasibility while reducing the execution time compared to kinodynamic motion planning. We selected a sampling-based path planning algorithm, namely, RRT* [1], which comes under geometric-based path planning that guarantees asymptotic optimality. On the other hand, RRT* may experience unpredictable performance issues due to its randomized behavior. Thus, there is no guarantee of the computational cost. In the proposed solution, we have a modified original RRT* [1] introducing an adaptive search space and a steering function that helps to increase the consistency of the planner. In addition, we propose a multiple RRT* that generates a set of desired paths where the optimal path is chosen based on a cost function. Finally, we apply kinodynamic smoothing, which will result in a dynamically feasible as well as obstacle-free path. Thereafter, a b-spline based trajectory is generated to maneuver the MAV autonomously.

Our Contributions:

1. Modifying the original RRT* algorithm introducing an adaptive search space and a steering function that helps to increase the consistency of the planner.
2. Proposing the planning of a horizon-based multiple RRT*. In each of the RRT* instances, it finds a feasible intermediate goal location and terminates at a newly generated goal position if the path length exceeds the planning horizon.
3. Applying iLQR (Iterative Linear Quadratic Regulator) smoothing to obtain the optimal obstacle-free path while ensuring the dynamic feasibility of MAV.

2. Related Work

2.1. Environment representation

Understanding the environment in real-time is important for collision-free planning. Hence, keeping track of the environment (map) close to the current position of MAV with a predefined perimeter is necessary provided that the map is updated in an incremental fashion. 3D voxel grid is one of the ways to represent the environment in a compact fashion. Each voxel grid stores information about its occupancy complemented with constant access time for retrieving any element. In spite of preceding advantages, it requires quite high memory-footprint. Hornung, et al. [2, 3] proposed Octomap, which uses probabilistic occupancy estimation to construct the map, namely, octree for storing the information. In the process of map building, random measurements, e.g., reflections, dynamics obstacles should be considered. Hence, environment representation with Octomap is concise and accurate enough for planning. Subsequently, voxel hashing [4] is

another technique, which uses Truncated Sign Distance Fields (TSDFs) [5] for reconstructing the environment.

2.2. Path planning

For path planning, a better understanding of the environment, as well as how environment is constructed and updated over time, is equally important. Initially, the path planning algorithm assumes an inaccessible area as a traversable area. Once MAV starts moving towards the desired direction, MAV starts exploring the unseen area and a feasible trajectory must be reconstructed, followed by replanning. Path planning can be classified into several levels [6]. Deterministic and probabilistic is one of the ways to classify them. For a given search space, the same result is expected for deterministic approaches, i. e., A* and Dijkstra. Deterministic approaches have several drawbacks including the fact that the path satisfying real-time constraints is not planned effectively and that the complexity increases when the search space dimension is higher, i. e., 3D. In contrast to deterministic approaches, probabilistic based approaches (e. g., Probabilistic Road Mapping [7], Randomly exploring Random Trees [8]) overcome those constraints despite adding more computation footprint. RRT* (Rapidly-exploring Random Tree) [1] is one of the well-adapted path planning techniques that can be categorized under sampling-based techniques.

2.3. Kinodynamic path planning

Most of the trajectory planning algorithms perform path planning followed by feasible trajectory generations as a two-step pipeline. Such a procedure may be problematic when it comes to replanning in which the path planner is unaware of the vehicle's dynamics. To address this problem, kinodynamic based motion planning [9] is highly desirable, which ensures the dynamic feasibility. Deterministic path planning algorithms such as A*, as well as probabilistic path planning techniques (e. g., RRT*, BIT* [10], FMT* [11], etc), can be modified by adding kinodynamic capabilities. However, real-time kinodynamic motion planning is seen as an open problem due to high computational cost. Moreover, a path generated by the path planner should be smooth. Frequently, spline-based techniques are utilized for smoothing without considering vehicle dynamics and obstacle constraints. Such trajectories are often infeasible trajectories. Such an infeasible trajectory generation can be avoided by applying kinodynamic path planning algorithms. One of such algorithms is the kinodynamic RRT, which was proposed by LaValle. et al. [12] and is applied for a hovercraft in a simulated environment. The most challenging task of the kinodynamic RRT* is to design an appropriate steering function [1, 13]. It is responsible for taking the control inputs and the current state and the preceding state (parent state), and connecting to the next state while ensuring dynamic feasibility.

To generate optimal control policy for a given MAV, which is utilized in kinodynamic planning, various techniques have been proposed. Linear Quadratic Regulator (LQR) was suggested by Glassman and Tedrake [14] for the kinodynamic RRT planner to generate an optimal control policy. Alejandro Perez, Robert Platt Jr [13] extended the preceding idea for RRT* while linearizing the nonlinear system dynamics at each new sample point. Moreover, they used an infinite horizon LQR policy to obtain optimal control inputs. Instead of using an infinite horizon, Jur and Berg [15] proposed finite horizon iterative LQR and extended LQR smoothing techniques considering nonlinear dynamics and nonquadratic cost for optimal kinodynamic motion planning.

2.4. Path smoothing

Path planning yields a sequence of waypoints which are connected by a set of straight lines and sharp turns. Hence, a path may not be desirable for navigation due to three constraints: geometric continuity, safety and dynamic feasibility. To fulfill these constraints, the path is

smoothed by ensuring those constraints. In general, there are three types of smoothing techniques: interpolate based, special curve based and optimization based. Polynomial interpolation, cubic splines and b-spline are good examples of interpolate based techniques. Dubin's curve and clothoids are some of the special curves which can be used for smoothing. However, none of those two technique takes into account dynamic feasibility and safety. Optimization-based approaches can satisfy all three preceding constraints. The authors of [16] proposed a convex elastic smoothing (CES) algorithm for trajectory smoothing as a convex optimization problem. Timed Elastic Band (TEB Planner) [17] is a kinodynamic planner which locally optimizes the trajectory while considering provided constraints. Zhou et al. [18] propose a kinodynamic local planner based on A*, which works quite aggressively.

2.5. Trajectory generation

There are several approaches for trajectory generation, including path planning followed by smoothing [19], motion primitive [20, 21] based, optimization-based [22, 23], etc. Path planning followed by smoothing is one of the widely adapted techniques among the preceding techniques. As explained in the previous section, the output of the path planner will be a sequence of waypoints in SO(3) that connects the start and goal positions coarsely. Such waypoints constitute the input to the trajectory generator. The trajectory generator approximates the desired path with time-based waypoints, which consists of position, velocity, acceleration, yaw, pitch and roll angles in SO(3) for a given time index. The newly generated trajectory must be feasible. If the generated trajectory satisfies all the limitations and constraints that are imposed by the dynamics of the vehicle and constraints on the environment, such a trajectory can be regarded as a feasible trajectory.

B-spline [24], minimum-snap [25] and its variants have widely been used for trajectory generation in the recent past. The initial stage of trajectory generation related work linearizes the model around the hover conditions which cannot be stabilized for reasonably high roll and pitch angles [26]. Minimum-snap trajectory generator is one of the first introduced techniques for trajectory generation in the tightly constrained setting. Minimum-snap can work with non-linear controllers while satisfying provided constraints (e.g., velocities, acceleration, input, etc) and ensuring the safe trajectory control sequences with respect to time. Besides, Minimum-snap is suitable for real-time based applications. On the other hand, spline functions, i.e., B-spline [27, 28], are extensively used for trajectory generation for several reasons. Clear geometric representation is very important for trajectory generation in 3D space. B-splines are much simpler from the computational point of view because local changes in the trajectory can be made quickly and easily without recomputing the entire trajectory [29]. Furthermore, if it is required to get derivatives higher than the second order (e.g., jerk, snap), this can be done by changing the order of the B-splines [30].

3. Methodology

The proposed framework is depicted in Fig. 1, whereas the workflow is shown in Fig. 2. The following sections explain in detail how each of the components contributes to acceptable system functioning.

3.1. Environment Representation

Since we use a depth camera for understanding the environment, it is necessary to build an incremental map because the camera has a limited field of view (FoV), i.e., $(85.2' \times 58' \times 94')$, which is not enough for planning. Thus, we feed the camera depth map into the Octomap server. The Octomap server constructs the map of the environment incrementally. Thereafter, the point

Fig. 1. Initially, the vehicle stays in the wait state until the goal position is provided. If the goal position is within the map and there are no obstacles at the goal position, its state will change into Gen, which will find an optimal trajectory between the current position and the goal position. Afterwards, the state will change into the Exec state until it reaches the goal position. If there are obstacles close to the vehicle, it retries to generate a new trajectory. This process continues until MAV navigates to the goal position. Besides, Exec state can be changed into Wait state if there are some sensor malfunctions that caused the sensor streams interruption. The PD (P-proportional, D-derivative) regulator takes actual and desired odometries and calculates the estimated velocity and yaw angle for controlling the vehicle

Fig. 2. The sequence of steps of the proposed framework. The depth camera works at 6 Hz, whereas the tracking camera works at 20 Hz. The depth map and odometry constitute an input for the Octomap server, which builds the map incrementally. Based on the output of the Octomap server, the instance map is constructed, which is utilized by Multiple RRT* for generating an optimal obstacle-free path. Afterwards, iLQR smoothing followed by trajectory generation is applied to the optimal path. Finally, the PD regulator is employed to synchronize the generated and actual trajectories

cloud around the current position of the MAV is extracted. RTree [31] is constructed from an extracted point cloud which represents the instance map of the environment.

3.2. Adaptive search space

RRT* is a well-known technique for path planning in high-dimensional spaces, i.e., 3D, in which the search space defines the whole map of the environment in the default setting. We

Algorithm 1: Generating points over the interior of the search space

```

1 input data:  $n$  points to be generated,  $\mathbf{r}$  principal semiaxes,  $R$  rotation matrix,  $c$  center position
2  $cols = 0, n_p = 0, r_{\min} = \min(\mathbf{r}), M = zero(3, n)$ 
3 if  $r_{\min} == \mathbf{r}_x$  then
4    $h = 2 \cdot \frac{\mathbf{r}_x}{2n+1}, n_i = n, n_j = \frac{\mathbf{r}_y}{\mathbf{r}_x} \cdot n, n_k = \frac{\mathbf{r}_z}{\mathbf{r}_x} \cdot n$ 
5 end
6 if  $r_{\min} == \mathbf{r}_y$  then
7    $h = 2 \cdot \frac{\mathbf{r}_y}{2n+1}, n_j = n, n_i = \frac{\mathbf{r}_x}{\mathbf{r}_y} \cdot n, n_k = \frac{\mathbf{r}_z}{\mathbf{r}_y} \cdot n$ 
8 end
9 else
10   $h = 2 \cdot \frac{\mathbf{r}_z}{2n+1}, n_k = n, n_i = \frac{\mathbf{r}_x}{\mathbf{r}_z} \cdot n, n_j = \frac{\mathbf{r}_y}{\mathbf{r}_z} \cdot n$ 
11 end
12 for  $k = 0, \dots, n_k$  do
13    $z = c_z + k \cdot h$ 
14   for  $j = 0, \dots, n_j$  do
15      $y = c_y + j \cdot h$ 
16     for  $i = 0, \dots, n_i$  do
17        $x = c_x + i \cdot h$ 
18        $M(0) = \langle x, y, z \rangle$ 
19       for  $l = 0, 1, 2, \dots, 7$  do
20         if  $l == 0 \parallel l == 1 \parallel l == 4$  then
21            $n_p = 0$ 
22         end
23         if  $l \leq 0$  then
24            $M(cols + l) = \langle 2c_x - M(n_p)_x, M(n_p)_y, M(n_p)_z \rangle \cdot R$ 
25         end
26         if  $1 \leq l \leq 2$  then
27            $M(cols + l) = \langle M(n_p)_x, 2c_y - M(n_p)_y, M(n_p)_z \rangle \cdot R$ 
28         end
29         if  $l \geq 3$  then
30            $M(cols + l) = \langle M(n_p)_x, M(n_p)_y, 2c_z - M(n_p)_z \rangle \cdot R$ 
31         end
32          $n_p++$ 
33          $cols++ = 8$ 
34       end
35     end
36   end
37 end

```

have made several changes to reduce the execution time and increase the consistency. In most situations, RRT* is utilized for global path planning, not for local planning, especially replanning. On the contrary, in this study, we use RRT* for local replanning. Thus, ensuring consistency of consecutive paths is required. One way to improve consistency is to generate proper random samples and then to define the optimal local search space which is closer to the current position of the vehicle. Considering the preceding facts, we proposed a deterministic way of generating search space, which eventually helps to improve the consistency of the planner significantly. In spite of having the capability of generating an asymptotically optimal path, it might not be consistent with the preceding path.

Search space is constructed as an ellipsoidal search space (spherical or oblate spheroid or prolate spheroid) whose principal axes are given by the start and goal positions of the trajectory. In general, random samples can be generated within the constructed ellipsoid, but we propose a deterministic way of generating sample points (Algorithm 1) similar to¹ (see Fig. 3) while considering constraints of the traversable space. Such a deterministic representation helps to reduce the execution time. Since search space is changed according to the traversable space around the current position of MAV, we call the search space an *adaptive search space*. The RRT* planner picks points randomly from the adaptive search space.

The adaptive search space is defined by

$$\frac{(x - c_x)^2}{(\mathbf{r}_x)^2} + \frac{(y - c_y)^2}{(\mathbf{r}_y)^2} + \frac{(z - c_z)^2}{(\mathbf{r}_z)^2} = 1, \quad (3.1)$$

where c is the middle position between the start position $\mathbf{x}_{start}$ and the goal position $\mathbf{x}_{goal}$ of the path and $\mathbf{r} = \langle \mathbf{r}_x, \mathbf{r}_y, \mathbf{r}_z \rangle$ is determined by

$$\mathbf{r} = \mathbf{x}_{goal} - \mathbf{x}_{start}, \quad (3.2)$$

where $\mathbf{r}_x = \max(4.0, \mathbf{r}_x)$, $\mathbf{r}_y = \max(4.0, \mathbf{r}_y)$ and $\mathbf{r}_z = \max(4.0, \mathbf{r}_z)$; the values of $\mathbf{r}_x$, $\mathbf{r}_y$, and $\mathbf{r}_z$ have been set to 4.0 m at max, assuming that the maximum area which covers the search space is sufficient to have 4.0 m at max in each direction. Once $\mathbf{r}$ is obtained, the search space orientation is rotated around $\mathbf{r}$. For the procedure of obtaining such a rotation matrix R , we have used a method that is proposed in [32].

3.3. Multiple RRT*

We have used multiple RRT* instances (see Fig. 3) using a thread pool that utilizes multi-cores/multiprocessors, resulting in N number of paths being generated, to further improve the consistency of path planning. N is a configurable parameter. It is better to select N as the number of cores in the embedded computer because the thread pool is created in the phase of algorithm initialization with the number of threads that is equal to the number of system cores, which helps to improve performance.

Initially, select a path that has the lowest cost as the optimal path based on the following cost:

$$Cost = \|(\mathbf{x}_M - \mathbf{x}_{goal})\| + \sum_{m=1}^{M-1} \|(\mathbf{x}_m - \mathbf{x}_{m+1})\| \quad (3.3)$$

where $\mathbf{x}_m$ depicts the m th waypoint of the selected path in which it consists of M number of waypoints. It should be mentioned that cost is not the only factor that should be considered when selecting a path. We have to check the safety of the path. If the chosen path is closer to obstacles, next time it will pick the path which has the second-lowest-cost. If the environment is cluttered, the optimal selection will go up to the highest cost as well. Once an optimal path is selected and the MAV starts moving on the trajectory, next time it will start from the lowest cost. If the distance between the start and goal positions is larger than a predefined value (planning horizon), RRT* will return the path up to the planning horizon. This is the procedure for selecting an optimal path from the generated paths.

¹Ellipsoid fitting <https://people.sc.fsu.edu>

Fig. 3. Multiple RRT* can generate a given number of paths between the start and goal positions (i.e., four paths) in this scenario

3.4. Path smoothing via iLQR

An optimal path that is chosen from the preceding step is smoothed while considering obstacles around the MAV. The optimal path consists of a set of waypoints that connect the start position and the goal position by a set of straight-lines. Afterwards, take each consecutive three waypoints, and get midpoints between the first and second waypoints, and the second and third waypoints and apply iLQR between those two midpoints. iLQR is solved as a finite horizon (N steps) optimization problem. The N number of steps at max is used to smooth the path.

We have used the same MAV model [33]. Let the system state space be $\mathbf{x} = [p^T v^T r^T w^T] \in \mathbb{R}^{n_x}$, where p , v , r and w stand for the position (m), velocity (m/s), the orientation about the $\mathbf{r}$ axis by angle $|r|$ (rad) and the angular velocity (rad/s), respectively, in $\mathbb{R}^3$. The size of the state space ($\mathbf{x}$) is equal to twelve (n_x) and the system has four (n_u) control inputs: $\mathbf{u} = [u_1, u_2, u_3, u_4]$. The MAV continuous-time dynamics $\dot{\mathbf{x}} = f_c(\mathbf{x}(t), \mathbf{u}(t))$ is described by

$$\begin{aligned} \dot{p} &= v, \\ \dot{v} &= -ge_3 + \frac{(u_1 - u_2 + u_3 + u_4) \exp([r]) * e_3 - k_v v}{m}, \\ \dot{r} &= w + \frac{1}{2}[r]w + \frac{\left(1 - \frac{|r|}{2 \tan(\frac{1}{2}|r|)}\right) [r]^2 w}{|r|^2}, \\ \dot{w} &= J^{-1}(\rho(u_2 - u_4)e_1 + \rho(u_3 - u_1)e_2 + k_m(u_1 - u_2 + u_3 - u_4)e_3 - [w])Jw, \end{aligned} \quad (3.4)$$

where e_i ; $i = 1, 2, 3$ denotes the basis of the local coordinate frame, $g = 9.8 \text{ m/s}^2$ stands for gravity, $k_v = 0.18$ and $k_m = 0.020$ are constants, $m = 0.67 \text{ kg}$, $J = 0.056I \text{ (kg m}^2\text{)}$, and $\rho = 0.017 \text{ m}$ are the mass, moment of inertia, and the distance from the center of the vehicle to the center of the rotor, respectively. $[a]$ depicts the skew-symmetric matrix notation. Afterwards, it is needed to define an optimal control policy that is utilized for generating a path between two given positions while ensuring the MAV dynamics and environmental constraints. Since the

continuous-time system dynamics is known, the discrete-time system dynamics can be formulated as

$$\mathbf{x}(t+1) = f_d(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t) \in \mathbb{R}^{n_x}, \quad \mathbf{u}(t) \in \mathbb{R}^{n_u}, \quad (3.5)$$

where $f_d(\mathbf{x}(t), \mathbf{u}(t))$ is determined by the 4th-order Runge–Kutta integrator as follows:

$$\begin{aligned} f_d(\mathbf{x}(t), \mathbf{u}(t)) &= \mathbf{x}(t) + \frac{dt}{6.0} \cdot (k1 + 2.0 \cdot k2 + 2.0 \cdot k3 + k4), \\ k1 &= f_c(\mathbf{x}(t), \mathbf{u}(t)), \\ k2 &= f_c(\mathbf{x}(t) + 0.5 \cdot dt \cdot k1, \mathbf{u}(t)), \\ k3 &= f_c(\mathbf{x}(t) + 0.5 \cdot dt \cdot k2, \mathbf{u}(t)), \\ k4 &= f_c(\mathbf{x}(t) + dt \cdot k3, \mathbf{u}(t)), \end{aligned} \quad (3.6)$$

where dt is set to 0.1 s. To obtain an optimal control policy, the following control objective (3.7) is minimized:

$$\sum_{k=0}^{N-1} c_t(\mathbf{x}_k(t), \mathbf{u}_k(t)) + c_N(\mathbf{x}_N(t)), \quad (3.7)$$

where c_t and c_N correspond to the local and final cost, respectively. Then the optimal control policy $(\pi_t(\mathbf{x}(t)) = [\mathbf{u}_0^*, \mathbf{u}_1^*, \dots, \mathbf{u}_{N-1}^*])$ can be calculated as follows:

$$\begin{aligned} &\text{minimize} \quad \sum_{k=0}^{N-1} c_t(\mathbf{x}_k(t), \mathbf{u}_k(t)) + c_N(\mathbf{x}_N(t)) \\ &\mathbf{u}_0, \dots, \mathbf{u}_{N-1} \end{aligned} \quad (3.8a)$$

$$\text{subject to} \quad \mathbf{x}_1(t) = f_d(\mathbf{x}_0(t), \mathbf{u}_0(t)), \quad (3.8b)$$

$$\mathbf{x}_2(t) = f_d(\mathbf{x}_1(t), \mathbf{u}_1(t)), \quad (3.8c)$$

$$\dots, \quad (3.8d)$$

$$\mathbf{x}_N(t) = f_d(\mathbf{x}_{N-1}(t), \mathbf{u}_{N-1}(t)), \quad (3.8e)$$

where $\mathbf{x}_0(t)$ is the starting position where the $\pi_t(\mathbf{x}(t))$ is calculated. iLQR provides a close-form solution for (3.8a). In iLQR, c_t and c_N are determined as follows:

$$\begin{aligned} c_t(\mathbf{x}(t), \mathbf{u}(t)) &= \frac{1}{2} \underbrace{\begin{bmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \end{bmatrix}}_{\mathbf{v}(t)}^T \underbrace{\begin{bmatrix} Q_t & P_t^T \\ P_t & R_t \end{bmatrix}}_{M_t} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \end{bmatrix} + \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \end{bmatrix}^T \underbrace{\begin{bmatrix} q_t \\ r_t \end{bmatrix}}_{m_t}, \\ c_N(\mathbf{x}(t)) &= \frac{1}{2} \mathbf{x}_N(t)^T Q_N \mathbf{x}_N(t) + \mathbf{x}_N(t)^T q_N \end{aligned} \quad (3.9)$$

where $Q_t \in \mathbb{R}^{n_x \times n_x}$, $R_t \in \mathbb{R}^{n_u \times n_u}$, $P_t \in \mathbb{R}^{n_u \times n_x}$, $q_t \in \mathbb{R}^{n_x}$ and $r_t \in \mathbb{R}^{n_u}$ and $\begin{bmatrix} Q_t & P_t^T \\ P_t & R_t \end{bmatrix} \geq 0$ in which Q_t and R_t are positive-definite matrices [15]. Knowing the cost estimation (3.9), discrete-dynamics (3.5) can be rewritten as follows:

$$\mathbf{x}(t+1) = f_d(\mathbf{x}(t), \mathbf{u}(t)) = A_t \mathbf{x}(t) + B_t \mathbf{u}(t) + c_t(\mathbf{x}(t), \mathbf{u}(t)) \quad \forall t, \quad (3.10)$$

where $A_t \in \mathbb{R}^{n_x \times n_x}$, $B_t \in \mathbb{R}^{n_x \times n_u}$, and $c_t \in \mathbb{R}^{n_x}$. Since the MAV dynamics is nonlinear at each considered discrete time instance t , the quadrotor dynamics $f_c(\mathbf{x}(t), \mathbf{u}(t))$ is linearized (3.11) and we obtain A_t and B_t :

$$A_t = \frac{\partial f_c(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{x}(t)}, \quad B_t = \frac{\partial f_c(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)}. \quad (3.11)$$

We can redefine (3.9) in the following way:

$$c_t(\mathbf{x}(t), \mathbf{u}(t)) = \frac{1}{2} \mathbf{v}(t)^T M_t \mathbf{v}(t) + \mathbf{v}(t)^T m_t. \quad (3.12)$$

To apply iLQR, it is required to provide M_N and m_N for the terminal state (the end state of horizon N). Then $c_t(\mathbf{x}(t), \mathbf{u}(t))$ ($M_t \in \mathbb{R}^{n_x \times n_x} > 0$ and $m_t \in \mathbb{R}^{n_x}$ are computed by using backward recursion with the help of the Riccati equation [34] for $N - 1, \dots, 0$. For example,

$$\begin{aligned} M_t &= Q_t + A_t^T M_{t+1} A_t - (P_t + B_t^T M_{t+1} A_t)^T (R_t + B_t^T S_{t+1} B_t)^{-1} (P_t + B_t^T M_{t+1} A_t), \\ m_t &= q_t + A_t^T m_{t+1} + A_t^T M_{t+1} c_t - (P_t + B_t^T M_{t+1} A_t)^T (R_t + B_t^T M_{t+1} B_t)^{-1} e_t, \end{aligned} \quad (3.13)$$

where $e_t = r_t + B_t^T m_{t+1} + B_t^T M_{t+1} c_t$. Thus, the optimal control policy $\pi_t(\mathbf{x}(t))$ has the following linear form:

$$\pi_t(\mathbf{x}(t)) = -(R_t + B_t^T M_{t+1} B_t)^{-1} (P_t + B_t^T M_{t+1} A_t) x_t - (R_t + B_t^T M_{t+1} B_t)^{-1} e_t. \quad (3.14)$$

In this way, M_t and m_t can be calculated starting from $t = N - 1$ until $t = 0$. Besides, obstacles cost $\sum_i \exp(-d_i(\mathbf{x}(t)))$ are being added into c_t as proposed in [15], where $d_i(\mathbf{x}(t))$ is the sign distance to the center of each obstacle. d_i is calculated with the help of RTree by retrieving close obstacles to the projected trajectory at each time step considered. Afterwards, we applied the iLQR smoothing technique between consecutive midpoints of two adjacent waypoints that are generated from RRT*. An example scenario is shown in Fig 4. Finally, the b-spline trajectory is generated using the smoothed path in which the MAV is maneuvered safely.

Fig. 4. The proposed multiple RRT* is capable of selecting an optimal path out of the generated paths. Then the optimal path is smoothed by iLQR smoothing

3.5. B-Spline Trajectory Generation

Multiple RRT* followed by iLQR smoothing gives a sequence of waypoints (n_s number of waypoints), which are guaranteed to be in the obstacle-free zone. Let those waypoints be the control points $(\mathbf{c}_i)_{i=1}^{n_s} \in \mathbb{R}^3$ for a uniform b-spline curve $f_b(t)$, which is a function of time, with a degree d (in our case, d equals 3). To generate the knot vector $(t_i)_{i=1}^{n_s+d}$, we set the step size ($step_size$) as 0.1 s, then the knot vector is calculated as follows:

$$t_{i=0}^{n_s+d+1} = \begin{cases} (-d+i) \cdot step_size, & t \leq d, \\ t_{i-1} + step_size, & t > d. \end{cases}$$

Knowing the control points and the knot vector, we can define $f_b(t)$ by

$$f_b(t) = \sum_{i=d+1}^n \mathbf{c}_i B_{i,d}(t), \quad (3.15)$$

where $B_{i,d}$ is given as

$$B_{i,d}(t) = \frac{t - t_i}{t_{i+d} - t_i} B_{i,d-1}(t) + \frac{t_{i+d+1} - t}{t_{i+d+1} - t_{i+1}} B_{i+1,d-1}(t) \quad (3.16)$$

with

$$B_{i,0}(t) = \begin{cases} 1, & t_i < t < t_{i+1}, \\ 0, & \text{otherwise.} \end{cases} \quad (3.17)$$

The derivative of B-spline is also a b-spline whose order becomes $d-1$. Let $\dot{\mathbf{c}}_i$ be the i th derivative of control points $\mathbf{c}_i$ which is defined by

$$\dot{\mathbf{c}}_i = \frac{\mathbf{c}_{i+1} - \mathbf{c}_i}{t_{i+d+1} - t_{i+1}}. \quad (3.18)$$

In this way, both velocity $\mathbf{v}_t \in \mathbb{R}^3$ and acceleration $\mathbf{a}_t \in \mathbb{R}^3$ components are calculated for a time index t that yields the desired MAV control input. The calculation of the desired yaw angle α_t is relaxed when generating a trajectory using b-spline; this is due to MAV yaw angle changes when avoiding obstacles. Thus, the desired rate of yaw angle is estimated as follows.

Algorithm 2: Desired yaw angle estimation

1 **input:** $\mathbf{v}_t$ current velocity
 2 **output:** $\dot{\alpha}_t$ desired yaw rate
 3 $\mathbf{v} = \frac{\mathbf{v}_t}{\|\mathbf{v}_t\|}$
 4 $\dot{\alpha}_t = \begin{cases} \text{atan2}(\mathbf{v}_y, \mathbf{v}_x), & \mathbf{v}_y \geq 0, \\ \text{atan2}(\mathbf{v}_y, \mathbf{v}_x) + 2\pi, & \text{otherwise} \end{cases}$

3.6. PD regulator

The PD regulator is employed for synchronizing the desired position from the planner and the actual MAV position. Since we used a px4 controller² on MAV, one way to control the quadrotor is by sending the desired velocity $\mathbf{v}_t$ and the yaw rate $\dot{\alpha}_t$. Hence, the expected velocity and the yaw rate are estimated as given in Algorithm 3, where $k_p = 0.8$, $k_d = 0.27$, and $k_a = 2.5$ are PD regulation parameters and $\mathbf{x}_\mu^*$, $\mathbf{v}_\mu^*$; $\mu = x, y, z$; $* = a, d$ gives both the current position and the velocity along each axis to the actual and the desired trajectories, respectively. When the expected velocity exceeds its upper ($v_{\max}$) or lower ($v_{\min}$) boundaries, the expected value is clipped to the respective boundary value.

²px4 controller <https://dev.px4.io/v1.9.0>

Algorithm 3: PD regulator

```

1 input:  $\mathbf{x}^a \in \mathbb{R}^3$  actual position,  $\mathbf{x}^d \in \mathbb{R}^3$  desired position,  $\dot{\alpha} \in \mathbb{R}$  desired yaw rate,  $\dot{\alpha}^a \in \mathbb{R}$  actual
   yaw rate,  $\mathbf{v} \in \mathbb{R}^3$  actual velocity
2 output:  $\bar{\mathbf{v}} \in \mathbb{R}^3$  estimated velocity,  $\bar{\alpha} \in \mathbb{R}$  estimated yaw rate
3  $\delta \mathbf{x} = \mathbf{x}^a - \mathbf{x}^d$ 
4  $\bar{\mathbf{v}}_x = \text{clip}(\delta \mathbf{x}_x \cdot k_p - \mathbf{v}_x \cdot k_d, v_{\min}, v_{\max})$ 
5  $\bar{\mathbf{v}}_y = \text{clip}(\delta \mathbf{x}_y \cdot k_p - \mathbf{v}_y \cdot k_d, v_{\min}, v_{\max})$ 
6  $\bar{\mathbf{v}}_z = \text{clip}(\delta \mathbf{x}_z \cdot k_p - \mathbf{v}_z \cdot k_d, v_{\min}, v_{\max})$ 
7  $\bar{\alpha} = \dot{\alpha}_t - \dot{\alpha}_t^a$ 
8  $\bar{\alpha} = \begin{cases} - = 2\pi, & \bar{\alpha} > \pi, \\ + = 2\pi, & \bar{\alpha} < -\pi \end{cases}$ 
9  $\bar{\alpha} = k_a \cdot \bar{\alpha},$ 

```

Fig. 5. Comparison of the map updating time (ms) per point cloud scan. The range of the map set for 4 m around the quadcopter current position. Subplots (a), (b), (c) show the updating time as histogram

4. Experimental Results

The sparse matrix library, namely, Eigen, was used to implement the proposed planner using C++11 while enabling the GNU C++ compiler optimization level to -O2. All the simulated experiments were conducted on a computer with (Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz CPU and 8 GB RAM). In this section, we present a qualitative and quantitative comparison of the proposed local replanner. Firstly, we compare our proposed instance map building with two other techniques, i. e., 3D Ring Buffer (Fig. 5a), ESDF (Fig. 5b). Secondly, the performance of the improved RRT* is compared with A* and the original RRT*. Thirdly, the complete system is evaluated in various simulated environments.

The proposed instance map, which is using the RTree [31] method, is constructed when the depth map is available at 6 Hz. We have adopted this approach to reduce the execution time; the approach is used for map building on the MAV. In the proposed approach, it is necessary to keep two instance maps (current and previous); first, the previous map is utilized by the planner, and once the current map is built, the previous map is replaced with the current map. The main advantage of the proposed map building in comparison with the above-mentioned methods [18, 24] is that, despite building an incremental map, the proposed approach will save computational costs and execution time while maintaining an accuracy similar to the above-mentioned approaches.

We have compared the mean computation time of the improved RRT* with the original RRT* and A*; further we calculated the mean computation time of the main five operations that are involved in the proposed planner as percentage values, as shown in Fig. 6. All three

Fig. 6. Mean computation times of the main four operations involved in the proposed planner as percentages

algorithms utilize the same search space that consists of 50 random obstacles within a cube of 20 m. We have generated 100 different search spaces and calculated the mean computation time (Table 1). We were able to reduce the computation time considerably. Besides, the improved RRT* has a constrained search space, it helps to improve the consistency of the path finding as we detailed in the Methodology section.

Table 1. Mean computation time for path finding under which same search space utilized for 100 different trials

	A*	Original RRT*	Improved RRT*
Mean computation time (s)	1.9612	1.2512	0.9621

Finally, we have conducted a few experiments in different simulated environments. Figure 7a shows one of the selected simulated environments for testing the behavior of the planner. When the distance between the goal and start positions is less than the predefined horizon (Fig. 7b), it generates a complete path, otherwise it generates a path up to the horizon as shown in Fig. 7c. Hence, the proposed planner works well under different environment conditions.

Fig. 7. An improved RRT* generates an N number (e.g., 4) of paths and selects the optimal path. Afterwards, iLQR smoothing is applied on the selected path that is shown in orange color. The other paths are shown in blue, dark blue and green, respectively, whereas the small red-colored sphere is the goal position. (a) A sample environment that is used for testing the proposed replanner. (b) When the Euclidean distance between the start and goal positions is less than the prediction horizon of the RRT*, it will generate a full path between the start and goal positions (c) When the Euclidean distance between the start and goal positions is higher than the horizon of the RRT*, it will generate a path between the start and the horizon as shown here. The nominal path is reprojected whenever close obstacles are detected

5. Conclusion

We have presented a framework for path planning which is followed by kinodynamic smoothing while ensuring the vehicle dynamics feasibility for MAVs. We have chosen “RRT*” for path planning in which several improvements have been done to reduce the planning time, which helps to employ RRT* as a local path finding algorithm. Kinodynamic smoothing ensures the dynamic feasibility and an obstacle-free path. Finally, we have validated the proposed framework in various simulated environments to check the behavior of the planner. Our validation result shows the performance of the planner. In our future work, we are going to improve the efficiency of the path planner.

References

- [1] Karaman, S. and Frazzoli, E., Sampling-Based Algorithms for Optimal Motion Planning, *Int. J. Robot. Res.*, 2011, vol. 30, no. 7, pp. 846–894.
- [2] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W., OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees, *Auton. Robot.*, 2013, vol. 34, no. 3, pp. 189–206.
- [3] Steinbrücker, F., Sturm, J., and Cremers, D., Volumetric 3D Mapping in Real-Time on a CPU, in *IEEE Internat. Conf. on Robotics and Automation (ICRA'2014)*, pp. 2021–2028.
- [4] Nießner, M., Zollhöfer, M., Shahram, I., and Stamminger, M., Real-Time 3D Reconstruction at Scale Using Voxel Hashing, *ACM Trans. Graph.*, 2013, vol. 32, no. 6, Art. 169, 11 p.
- [5] Oleynikova, H., Taylor, Z., Fehr, M., Nieto, J., and Siegwart, R., Voxblox: Building 3D Signed Distance Fields for Planning, arXiv:1611.03631 (2016).
- [6] Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N., and Feyzeau, P., Path Planning: A 2013 Survey, in *Proc. of the 2013 Internat. Conf. on Industrial Engineering and Systems Management (IESM)*, 8 p.
- [7] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H., Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces, *IEEE Trans. Rob. Autom.*, 1996, vol. 12, no. 4, pp. 566–580.
- [8] LaValle, S. M., Rapidly-Exploring Random Trees: A New Tool for Path Planning, *Iowa State Univ. Technical Report 11*, 1998.
- [9] Donald, B., Xavier, P., Canny, J., and Reif, J., Kinodynamic Motion Planning, *J. ACM*, 1993, vol. 40, no. 5, pp. 1048–1066.
- [10] Lan, M., Lai, Sh., Bi, Y., Qin, H., Li, J., Lin, F., and Chen, B. M., BIT*-Based Path Planning for Micro Aerial Vehicles, in *IECON 2016: Proc. of the 42nd Annual Conf. of the IEEE Industrial Electronics Society*, pp. 6079–6084.
- [11] Starek, J. A., Gomez, J. V., Schmerling, E., Janson, L., Moreno, L., and Pavone, M., An Asymptotically-Optimal Sampling-Based Algorithm for Bi-Directional Motion Planning, in *IEEE/RSJ Internat. Conf. on Intelligent Robots and Systems (IROS'2015)*, pp. 2072–2078.
- [12] LaValle, S. M. and Kuffner, J. J., Jr., Randomized Kinodynamic Planning, *Int. J. Robot. Res.*, 2001, vol. 20, no. 5, pp. 378–400.
- [13] Perez, A., Platt, R., Konidaris, G., Kaelbling, L., and Lozano-Perez, T., LQR-RRT*: Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics, in *2012 IEEE Internat. Conf. on Robotics and Automation*, pp. 2537–2542.
- [14] Glassman, E. and Tedrake, R., A Quadratic Regulator-Based Heuristic for Rapidly Exploring State Space, in *2010 IEEE Internat. Conf. on Robotics and Automation*, pp. 5021–5028.
- [15] van den Berg, J., Extended LQR: Locally-Optimal Feedback Control for Systems with Non-Linear Dynamics and Non-Quadratic Cost in *Robotics Research*, M. Inaba, P. Corke (Eds.), Springer Tracts Adv. Robot., vol. 114, Cham: Springer, 2016, pp. 39–56.
- [16] Zhu, Zh., Schmerling, E., and Pavone, M., A Convex Optimization Approach to Smooth Trajectories for Motion Planning with Car-Like Robots, in *Proc. of the 54th IEEE Conf. on Decision and Control (CDC'2015)*, pp. 835–842.

- [17] Rösmann, Ch., Hoffmann, F., and Bertram, T., Kinodynamic Trajectory Optimization and Control for Car-Like Robots, in *IEEE/RSJ Internat. Conf. on Intelligent Robots and Systems (IROS'2017)*, pp. 5681–5686.
- [18] Zhou, B., Gao, F., Wang, L., Liu, Ch., and Shen, Sh., Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight, *IEEE Robot. Autom. Lett.*, 2019, vol. 4, no. 4, pp. 3529–3536.
- [19] Biagiotti, L. and Melchiorri, C., *Trajectory Planning for Automatic Machines and Robots*, Berlin: Springer, 2008.
- [20] Mueller, M. W., Hehn, M., and D'Andrea, R., A Computationally Efficient Algorithm for State-to-State Quadcopter Trajectory Generation and Feasibility Verification, in *2013 IEEE/RSJ Internat. Conf. on Intelligent Robots and Systems*, pp. 3480–3486.
- [21] Paranjape, A. A., Meier, K. C., Shi, X., Chung, S.-J., and Hutchinson, S., Motion Primitives and 3D Path Planning for Fast Flight through a Forest, *Int. J. Robot. Res.*, 2015, vol. 34, no. 3, pp. 357–377.
- [22] Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, Ch. M., Bagnell, J. A., and Srinivasa, S. S., Chomp: Covariant Hamiltonian Optimization for Motion Planning, *Int. J. Robot. Res.*, 2013, vol. 32, nos. 9–10, pp. 1164–1193.
- [23] Oleynikova, H., Burri, M., Taylor, Z., Nieto, J., Siegwart, R., and Galceran, E., Continuous-Time Trajectory Optimization for Online UAV Replanning, in *IEEE/RSJ Internat. Conf. on Intelligent Robots and Systems (IROS'2016)*, pp. 5332–5339.
- [24] Usenko, V., von Stumberg, L., Pangercic, A., and Cremers, D., Real-Time Trajectory Replanning for MAVs Using Uniform B-Splines and a 3D Circular Buffer, in *IEEE/RSJ Internat. Conf. on Intelligent Robots and Systems (IROS'2017)*, pp. 215–222.
- [25] Mellinger, D. and Kumar, V., Minimum Snap Trajectory Generation and Control for Quadrotors, in *2011 IEEE Internat. Conf. on Robotics and Automation*, pp. 2520–2525.
- [26] Hoffmann, G., Waslander, S., and Tomlin, C., Quadrotor Helicopter Trajectory Tracking Control, in *AIAA Guidance, Navigation and Control Conference and Exhibit (Honolulu, Hawaii, 2008)*, AIAA 2008-7410, 14 p.
- [27] Babaei, A. and Karimi, A., Optimal Trajectory-Planning of UAVs via B-Splines and Disjunctive Programming, arXiv:1807.02931 [math.OC] (2018).
- [28] Biagiotti, L. and Melchiorri, C., B-Spline Based Filters for Multi-Point Trajectories Planning, in *2010 IEEE Internat. Conf. on Robotics and Automation*, pp. 3065–3070.
- [29] Bobrow, J. E., Optimal Robot Plant Planning Using the Minimum-Time Criterion, *IEEE J. Robot. Autom.*, 1988, vol. 4, no. 4, pp. 443–450.
- [30] Wang, C.-H. and Horng, J.-G., Constrained Minimum-Time Path Planning for Robot Manipulators via Virtual Knots of the Cubic B-Spline Functions, *IEEE Trans. Automat. Control*, 1990, vol. 35, no. 5, pp. 573–577.
- [31] Guttman, A., R-Trees: A Dynamic Index Structure for Spatial Searching, in *Proc. of the 1984 ACM SIGMOD Internat. Conf. on Management of Data*, pp. 47–57.
- [32] Kulathunga, G., Fedorenko, R., Kopylov, S., and Klimchik, A., Real-Time Long Range Trajectory Replanning for MAVs in the Presence of Dynamic Obstacles, arXiv:2001.03605 (2020).
- [33] van den Berg, J., Iterated LQR Smoothing for Locally-Optimal Feedback Control of Systems with Non-Linear Dynamics and Non-Quadratic Cost, in *American Control Conference (Portland, OR, USA, June 2014)*, pp. 1912–1918.
- [34] *The Riccati Equation*, S. Bittanti, A. J. Laub, J. C. Willems (Eds.), Berlin: Springer, 2012.