Efficient Sampling-based Multirotors Kinodynamic Planning with Fast Regional Optimization and Post Refining

Hongkai Ye, Neng Pan, Qianhao Wang, Chao Xu and Fei Gao

***Abstract*— For real-time multirotor kinodynamic planning, the efficiency of sampling-based methods is usually hindered by difficult-to-sample homotopy classes like narrow passages. In this paper, we address this issue by a hybrid scheme. We firstly propose a fast regional optimizer exploiting the information of local environments and then integrate it into a bidirectional global sampling process. The incorporation of the local optimization shows significantly improved success rates and less planning time in various types of challenging environments. We further present a refinement module utilizing the same framework as the regional optimizer. It comprehensively investigates the resulting trajectory of the global sampling and improves its smoothness with nearly negligible computation effort. Benchmark results illustrate that our proposed method can better exploit a previous trajectory compared to the state-of-the-art ones. The planning methods are applied to generate trajectories for a quadrotor system in simulation and real-world, and their capability is validated in real-time applications.**

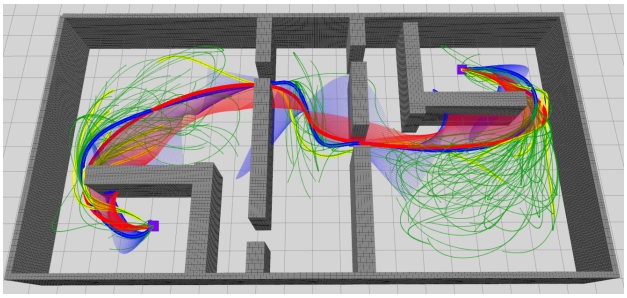


Fig. 1: Planning results in a maze-like trapped environment. The blue and red curves are the trajectories obtained by the proposed front-end and back-end, respectively, with the transparent lines indicating the accelerations. The yellow curves are some regionally optimized parts. The green curves show two growing trees.

I. Introduction

Multirotors can conduct agile maneuvers like catching a ball in the air [1], flying through narrow passages [2], or even acrobatics [3]. To enable such mobility and to prevent making motion plans that can not be fulfilled, the system dynamics, as well as the control and state saturation constraints, have to be considered when planning trajectories, which usually relates to kinodynamic motion planning [4].

Being asymptotically optimal and holding the anytime property, many sampling-based [5]–[7] kinodynamic variants of path planning methods have been successfully applied to solve the problem with various system settings. They seek to identify feasible motions in the continuous solution space by drawing discrete samples and build connections between them, which is commonly referred to as steering. Solving Boundary Value Problems (BVPs) is almost inevitable for steering, which is difficult, especially for nonlinear dynamics with a non-differentiable objective and complex constraints. Our previous work [8] eases this problem by relaxing constraints with linear models of multirotors. Still, it is not efficient to explore the whole solution space if difficult-to-sample homotopy classes like narrow passages exist, which can cost great efforts for a standalone sampling-based kinodynamic planner to sample through. Local trajectory optimization techniques can be of great help for these situations since they prioritize exploiting domain information to explore the local environment. As a result, integrating regional optimization (RO) into global exploration is embraced by many [9]–[13] and has become the go-to approach to deal with such problems. Empirically, this scheme's overall effect is dominated by the solving time of the local optimizer. Most works address nonlinear programming for the RO, and apply gradient descent to solve it. The explicit gradient computation, however, is not easy and not cheap to acquire, especially for complex obstacles. Unlikely, we think in a different manner and propose to formulate the RO as a series of unconstrained quadratic programmings (QP) with a closed-form solution for each iteration, which guarantees fast solution time and avoids the requirements of calculating any gradients.

To meet some real-time requirements, planning time budget is limited, and the sampling is often insufficient, causing the resulting trajectory of the aforementioned global kinodynamic planning not smooth enough with unpleasant motions to be directly fed to a vehicle, leaving space for refining. Many works [14]–[17] adopt a lightweight refinement module to preserve and improve some properties of the previous result, obtaining much better outcomes. This two-stage scheme is commonly known as the front-end & back-end framework of kinodynamic motion planning. The front-end plans a trajectory that satisfies all the constraints and settles in a proper homotopy class, whereas the back-end further efficiently refines the trajectory probably within the same homotopy class. It remains open to make full use of the front-end results while retaining efficiency, effectiveness,

All authors are with State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China and with Huzhou Institute of Zhejiang University, Huzhou 313000, China.

Corresponding author: Fei Gao.

Funding: This work was supported by the National Natural Science Foundation of China under Grant No. 62003299 and 62088101.

Email: (hkye, panneng_zju, qhwangaa, cxu and fgaoaa)@zju.edu.cn

Code available at: https://github.com/ZJU-FAST-Lab/kino_sampling_with_regional_opti

and high success rate.

In this paper, as an enhancement of our previous work [8] and focused on the gaps mentioned above that 1) narrow passages are difficult to address in the global sampling process, and 2) front-end assets are not fully exploited in the back-end refining, we present a kinodynamic planning method and framework that meets real-time requirements for multirotor flight. A variant of the kinodynamic RRT* (kRRT*) [7, 8] is adopted as the front-end to conduct the global search, and a new practical back-end is developed. Improvements are two-fold. Firstly, we integrate a fast regional trajectory optimizer and bidirectional search into the global reasoning process, greatly accelerating to find an initial solution with a higher success rate. Secondly, inheriting the ideas from our regional optimization, the back-end extends to consider the obstacle clearance in some degree. This refinement is extremely fast and has higher success rate than the previous ones.

The main contributions of this paper are:

1) We propose a fast trajectory optimization method as a sequence of QP based on [8] and [18], which obtains feasible solutions with much fewer iterations by taking obstacle clearance into account while a closed-form solution is preserved for each iteration.
2) We design a bidirectional RRT*-based kinodynamic planner integrated with the optimization method as a regional trajectory optimizer. Planning under this framework improves the initial solution time, success rate, and sample contribution rate of the global search.
3) We exploit the optimization method as a lightweight trajectory refinement back-end, which exploits the kinodynamic global planning assets and efficiently improves the resulting trajectory in smoothness and obstacle clearance.
4) We apply the proposed planning methods to trajectory generation of a quadrotor system, presenting extensive benchmarks and simulated as well as real-world experimental validations, and releasing source code for the reference of the community.

II. Related Works

A. Planning with Regional Optimization

Most standalone search-based and sampling-based global planning methods solely focus on exploring the entire solution space to bring an optimal answer. Local domain information is disregarded, and as a result, they suffer from low efficiency when narrow passages present. Regional optimizers, on the other hand, prioritize investigating a limited area thus are better with narrow spaces. Choudhury et al. [11] combine gradient-based local optimizers like CHOMP [19] with their global sampling process adopted from BIT* [20] and achieve faster access to an initial solution as well as a higher convergence rate in high dimensions. However, when few difficult-to-sample homotopy classes exist, the relatively heavy optimization can rather hinder the global process. Another disadvantage is that extra efforts are required to calculate the obstacle distance gradients a priori. Kim et al. [12, 13] follow similar schemes and develop a PRM-style [21] sparse graph to explore different homotopy classes. They instead get obstacle gradients with empirical collisions found during the execution. Though avoid prior computation, it is uneasy to obtain accurate gradients and thus impedes the convergence. For kinodynamic planning, Stoneman et al. [9], solve optimal control problems via sequential quadratic programming for the steering of their modified RRT*. Since all the constraints are directly considered, it takes seconds to solve and can only be used offline. Focused on multirotors planning, we take advantage of the differential flatness [22], and formulate the steering and regional optimization in closed-form. Constraints are checked afterwards.

B. Exploiting Front-end Result

Though all the constraints are satisfied, the trajectory obtained by the global reasoning part may lack smoothness due to the limited time budget. Therefore, a refinement module is usually followed to improve it. Some works [15, 17, 23] re-parameterize their front-end result in the back-end and form optimization problems with soft constraints, where a distance field or/and free corridors are obliged to provide obstacle clearance. Liu et al. [16] solve an unconstrained QP with intermediate waypoints and time allocation obtained from their search-based kinodynamic front-end and check feasibility afterwards. They do not need to build any time-consuming fields or corridors but require the intermediate waypoints fixed to provide collision-free information in the environments. Our previous work [8] relax this constraint and instead formulate a *Homotopy Cost* to force every point in the optimized trajectory to be close to the original collision-free one. However, no obstacle information is considered, and thus it takes many iterations to get a feasible result or even fail in complex environments. This work addresses it by further incorporating collision penalties found during each iteration on specific parts of the optimized trajectory.

III. Methodology

We begin by presenting the problem definition, and then introduce the modified kRRT* framework, highlighting the integrated differences, and then present the quadratic formulation and closed-form solution of the polynomial trajectory optimizer, and finally, the refinement back-end.

A. Preliminaries

With the help of the multirotor's differential flatness property [22], it allows us to use a linear model to represent its dynamics with four flat outputs p_x, p_y, p_z (position in each axis), ψ (yaw) and their derivatives for the state variables. To impose continuity in at least acceleration, we use the triple integrator model with jerk as the input:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \tag{1}$$

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}(t) \\ \dot{\mathbf{p}}(t) \\ \ddot{\mathbf{p}}(t) \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}, \tag{2}$$

$$\mathbf{u}(t) = \dddot{\mathbf{p}}(t), \quad \mathbf{p}(t) = \begin{bmatrix} p_x(t), & p_y(t), & p_z(t) \end{bmatrix}^\mathsf{T}.$$

Algorithm 1 KinoPlanning

Notation: Environment $\mathcal{E}$, Tree $\mathcal{T}$, State $\mathbf{x}$, Edge $\mathbf{e}$, Trajectory $\mathbf{T}$
Initialize: $\mathcal{T}_i \leftarrow \emptyset \cup \{\mathbf{x}_{init}\}$, $\mathcal{T}_g \leftarrow \emptyset \cup \{\mathbf{x}_{goal}\}$
for $i = 1$ to n **do**
  $\mathbf{x}_{random} \leftarrow$ **Sample**($\mathcal{E}$)
  $\mathcal{X}_{backward} \leftarrow$ **BackwardNear**($\mathcal{T}_i$, $\mathbf{x}_{random}$)
  $\mathbf{x}_i \leftarrow$ **ChooseParent**($\mathcal{X}_{backward}$, $\mathbf{x}_{random}$)
  $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{\mathbf{x}_i, \mathbf{x}_{random}\}$
  $\mathcal{X}_{forward} \leftarrow$ **ForwardNear**($\mathcal{T}_i$, $\mathbf{x}_{random}$)
  Rewire($\mathbf{x}_{random}$, $\mathcal{X}_{forward}$)
  $\mathcal{X}_{forward} \leftarrow$ **ForwardNear**($\mathcal{T}_g$, $\mathbf{x}_{random}$)
  $\mathbf{x}_g \leftarrow$ **ChooseParent**($\mathcal{X}_{forward}$, $\mathbf{x}_{random}$)
  $\mathcal{T}_g \leftarrow \mathcal{T}_g \cup \{\mathbf{x}_{random}, \mathbf{x}_g\}$
  $\mathcal{X}_{backward} \leftarrow$ **BackwardNear**($\mathcal{T}_g$, $\mathbf{x}_{random}$)
  Rewire($\mathbf{x}_{random}$, $\mathcal{X}_{backward}$)
  if $\mathbf{x}_i \bigwedge \mathbf{x}_g$ **then**
    one solution found
  end if
end for
$\mathbf{T} \leftarrow$ **RetriveTraj**()
Refine($\mathbf{T}$)

Following [8, 15, 16], we search for a trajectory starting from an initial state $\mathbf{x}_{init}$ to a goal state $\mathbf{x}_{goal}$ that is optimal in a trade-off between time and energy. The trajectory planning problem is formulated as follows:

$$\begin{aligned} \min_{\mathbf{x}(t)} \mathcal{J} &= \int_0^\tau (\rho + \frac{1}{2}\mathbf{u}(t)^\mathsf{T}\mathbf{u}(t))dt \\ s.t. \quad & \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) - \dot{\mathbf{x}}(t) = \mathbf{0}, \\ & \mathbf{x}(0) = \mathbf{x}_{init},\ \mathbf{x}(\tau) = \mathbf{x}_{goal}, \\ & \forall t \in [0, \tau],\ \mathbf{x}(t) \in \mathcal{X}^{free},\ \mathbf{u}(t) \in \mathcal{U}^{free}, \end{aligned} \tag{3}$$

where τ is the time duration of the trajectory and ρ the trade-off weight to penalize time against energy. $\mathcal{X}^{free}$ stands for states that are obstacle-free and satisfy derivative constraints. $\mathcal{U}^{free}$ indicates feasible controls.

Polynomial splines are widely used to express time and energy optimal trajectories in continuous-time motion planning owing to their strong manifestations with simple parameterization. We use polynomials as trajectory bases in both front-end and back-end, and the final solution trajectory is expressed as piece-wise polynomials. For differential flat systems, each flat output dimension can be decoupled and planed, respectively. For each dimension (yaw planning is excluded in this paper), consider an $(n+1)$-order, m-piece spline with its i^{th} segment an n-degree polynomial $p_i(t) = \mathbf{c}_i^\mathsf{T}\mathbf{t}, t \in [0, T_i]$, where $\mathbf{c}_i \in \mathbb{R}^{n+1}$ is the coefficient vector of the i^{th} segment, $\mathbf{t} = (1, t, t^2, ..., t^n)^\mathsf{T}$ the natural basis, and T_i the time duration.

Algorithm 2 ChooseParent($\mathcal{X}$, $\mathbf{x}$)

$\mathbf{x}_{min} \leftarrow null,\ min\_cost \leftarrow inf$
for $\mathbf{x}_{parent}$ in $\mathcal{X}$ **do**
  $\mathbf{e} \leftarrow$ **Steering**($\mathbf{x}_{parent}$, $\mathbf{x}$)
  if CheckFeasible($\mathbf{e}$, $\mathcal{E}$) $\bigwedge$ **Cost**($\mathbf{e}$) $< min\_cost$ **then**
    $\mathbf{x}_{min} \leftarrow \mathbf{x}_{parent},\ min\_cost \leftarrow$ **Cost**($\mathbf{e}$)
  else
    $\mathbf{T} \leftarrow$ **RegionalOptimize**($\mathbf{e}$)
    if CheckFeasible($\mathbf{T}$, $\mathcal{E}$) $\bigwedge$ **Cost**($\mathbf{T}$) $< min\_cost$ **then**
      $\mathbf{x}_{min} \leftarrow \mathbf{x}_{parent},\ min\_cost \leftarrow$ **Cost**($\mathbf{T}$)
    end if
  end if
end for
return $\mathbf{x}_{min}$

B. Bidirectional kRRT with Regional Optimization*

The main workflow of the Bidirectional kRRT* is described in the looping part of Alg. 1, where two trajectory trees $\mathcal{T}_i$ and $\mathcal{T}_g$ grow towards each other from the initial state $\mathbf{x}_{init}$ and the goal state $\mathbf{x}_{goal}$, respectively. In **ChooseParent()** and **Rewire()**, the **Steering()** that builds a connecting edge between two states acts as a fundamental unit, and requires the solving of a BVP described with Eq. 3. Each drawn sample tries to connect to both $\mathcal{T}_i$ and $\mathcal{T}_g$ through **ChooseParent()**. The difference is that when connecting to $\mathcal{T}_i$, the sample provides a final state for the BVP whereas it provides an initial state in the case of connecting to $\mathcal{T}_g$. If both connected, then a new trajectory is found. A visualization of the two growing trees is shown in Fig. 1.

Building on [8], we temporarily assume the state and control unbounded and solve it by Pontryagin Maximum Principle, obtaining the optimal transition time τ^* and deriving the unconstrained optimal transition trajectory as a 5^{th}-degree polynomial in several microseconds. This edge (a piece of trajectory) is then checked for constraints feasibility by **CheckFeasible()**. In [8], the edge is merely aborted if any violation occurs. In this paper, however, we reuse the unqualified edge by regional optimization to improve efficiency. If the derivative constraints are violated, we increase τ^* and recompute the polynomial coefficients until they are not. If the edge collides with any obstacle, it is checked for conditional regional optimization, as is presented in the following sections.

C. Quadratic Objective Formulation

Being an integrated part of the frequently called steer function, the optimization process needs to be computationally fast. Investigating the edge that is checked collided, it has a reasonable time duration, and we know where the collision occurs. It is desired to deform the trajectory to a collision-free one in its nearby free solution space with as little effort. Enhancing the works in [8, 18] where the objective is in quadratic form, and the matrix of the quadratic term is positive definite (PD) such that closed-form solutions

are available, we further add an objective term of collision in aware of the obstacle areas. The single edge is uniformly divided (**Split()** in Alg. 3) by time into j trajectory pieces of the same degree to bring in more freedom for the following optimization. For each axis out of x, y, and z, the quadratic objective consists of three terms:

1) Smoothness Cost: J_s is formulated as the time integral of the squared jerk of the trajectory:

$$\begin{aligned} J_s &= \int_0^T [p^{(3)}(t)]^2 dt \\ &= \sum_{i=1}^{j} \mathbf{c}_i^\mathsf{T} \int_0^{T_i} \mathbf{t}^{(3)} (\mathbf{t}^{(3)})^\mathsf{T} dt \ \mathbf{c}_i = \mathbf{c}^\mathsf{T} \mathbf{Q}_s \mathbf{c}, \end{aligned} \tag{4}$$

where $T = T_1 + T_2 + \cdots + T_j$ is the total duration of the trajectory and T_i the duration of one divided piece. $\mathbf{c}^\mathsf{T} = [\mathbf{c}_1^\mathsf{T}, \mathbf{c}_2^\mathsf{T}, \cdots, \mathbf{c}_j^\mathsf{T}]$ is the coefficient vector of the j segments.

2) Resemblance Cost: J_r is formulated as the integration over the squared difference between positions of the optimized trajectory and the originally divided trajectory $p^*(t)$:

$$\begin{aligned} J_r &= \int_0^T [p(t) - p^*(t)]^2 dt \\ &= \sum_{i=1}^{j} (\mathbf{c}_i - \mathbf{c}_i^*)^\mathsf{T} \int_0^{T_i} \mathbf{t}\mathbf{t}^\mathsf{T} dt \ (\mathbf{c}_i - \mathbf{c}_i^*) \\ &= (\mathbf{c} - \mathbf{c}^*)^\mathsf{T} \mathbf{Q}_r (\mathbf{c} - \mathbf{c}^*). \end{aligned} \tag{5}$$

This term drives the optimized trajectory to be close in position to the original one, and thus the collision-free parts are likely to remain feasible.

3) Collision Cost: J_c is formulated similar to the *Resemblance Term*. The difference is that the subtracted trajectory ($p^*(t)$) is now some selected attracting points ($p_{ap}(t)$) providing dragging force to draw the collided part of the trajectory to nearby collision-free areas.

$$\begin{aligned} J_c &= \sum_{ap \in APs} \int_{t_{s,ap}}^{t_{e,ap}} [p(t) - p_{ap}(t)]^2 dt \\ &= \sum_{ap \in APs} \sum_{i \in L} (\mathbf{c}_i - \mathbf{c}_i^{ap})^\mathsf{T} \int_{t_{s,i,ap}}^{t_{e,i,ap}} \mathbf{t}\mathbf{t}^\mathsf{T} dt \ (\mathbf{c}_i - \mathbf{c}_i^{ap}) \\ &= \sum_{ap \in APs} (\mathbf{c} - \mathbf{c}^{ap})^\mathsf{T} \mathbf{Q}_{c,ap} (\mathbf{c} - \mathbf{c}^{ap}), \end{aligned} \tag{6}$$

where $p_{ap}(t)$ is the constant position of one attracting point ap out of the set APs, and ($t_{e,ap}$ - $t_{s,ap}$) $\subseteq [0, T]$ is the corresponding time period of the optimized trajectory that is affected by the dragging force. L is the set of specific piece indices that is drawn by an attracting point with ($t_{e,i,ap}$ - $t_{s,i,ap}$) $\subseteq [0, T_i]$ the involved time period in the i^{th} piece.

This term is critical since it implicitly conveys the obstacle information and guides the deforming towards obstacle-free areas.

The overall objective is formed as a weighted sum of the three terms, and the optimization problem is formulated in the following:

$$\begin{aligned} \min J &= \lambda_s J_s + \lambda_r J_r + \lambda_c J_c \\ &= [\mathbf{c}^\mathsf{T} (\lambda_s \mathbf{Q}_s + \lambda_r \mathbf{Q}_r + \lambda_c \sum_{ap \in APs} \mathbf{Q}_{c,ap}) \mathbf{c} \\ &\quad - 2\mathbf{c}^\mathsf{T} (\lambda_r \mathbf{Q}_r \mathbf{c}^* + \lambda_c \sum_{ap \in APs} \mathbf{Q}_{c,ap} \mathbf{c}^{ap}) \\ &\quad + \lambda_r (\mathbf{c}^*)^\mathsf{T} \mathbf{Q}_r \mathbf{c}^* + \lambda_c \sum_{ap \in APs} (\mathbf{c}^{ap})^\mathsf{T} \mathbf{Q}_{c,ap} \mathbf{c}^{ap}] \\ s.t. \quad & \mathbf{A}\mathbf{c} = \mathbf{d}, \\ & \forall t \in [0, \tau], \ \mathbf{x}(t) \in \mathcal{X}^{free}, \ \mathbf{u}(t) \in \mathcal{U}^{free}, \end{aligned} \tag{7}$$

where λ_s, λ_r, and λ_c are the respective weights, c the decision variable vector, and $\mathbf{A}\mathbf{c} = \mathbf{d}$ the boundary derivative constraints for each piece.

In this way, we seek a smoother trajectory close to the original one for the collision-free parts while deforms to neighbor collision-free areas for the collided parts.

Algorithm 3 RegionalOptimize(e) / Refine(T)

```
T ← Split(e)
while iter_num < max_num do
    Adjust()
    T_temp ← ClosedFormSolve()
    if CheckFeasible(T_temp, E) then
        T ← T_temp
        break
    else
        Adjust()
    end if
end while
return T
```

D. Iterative Optimization Process

As we can see, the overall objective J is quadratic, and the coefficient matrix of the quadratic term is always positive definite if the weights and time durations are non-negative. Following [18, 24] and utilizing the PD property, an unconstrained formulation of QP incorporating the boundary derivative constraints can be derived. Ignoring the state and control constraints, the optimal solution can be obtained in closed-form with given boundary conditions and time allocation, which is the process of **ClosedFormSolve()** in Alg. 3. We then check for state and control saturation and obstacle feasibility of the unconstrained solution. If it collides with new obstacles, we reformulate the *Collision Cost* by incrementally adding new selected attracting points to provide more accurate dragging forces and information of the local surrounding environment, which is the process of **Adjust()** in Alg. 3. If the state and control violate saturations, we increase the time duration of the whole trajectory. This process runs iteratively until a feasible trajectory is found, or maximum iteration time is reached.

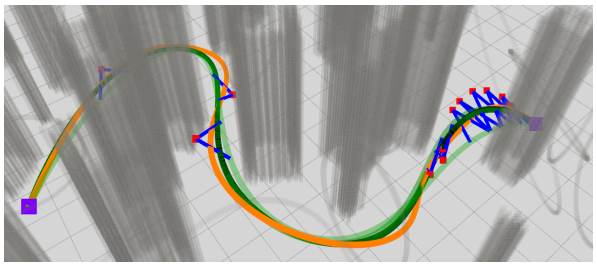
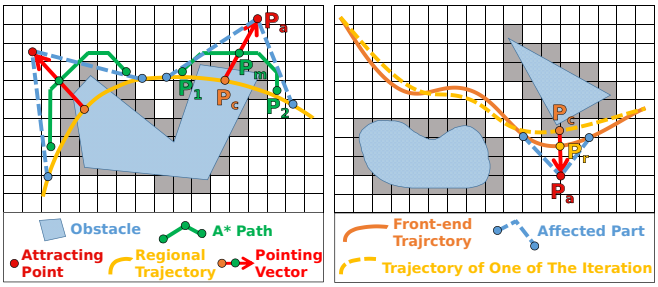


Fig. 2: Visualization of the iteration process of back-end optimization. Obstacles are set transparent to provide better views. The orange curve is the front-end trajectory. The red dots represent attracting points, with blue lines indicating the affected parts. As iterations proceed, the previous attracting points preserve while more are added, and the optimized trajectory is shown as green curves colored from light to dark.



(a) Regional Optimizer Cases (b) Back-end Cases

Fig. 3: Illustration of selecting attracting points in 2D, the method applies for 3D environments as well. (a) For the regional optimization cases, when the regional trajectory (yellow curve) collides with obstacles, we denote the start and endpoint of collision as P_1 and P_2, the point in the middle of the collision part as P_c. Then we search a free path (green lines) from P_1 to P_2 and denote the middle point in the path as P_m. The attracting point P_a is chosen in some distance in the extending line of the pointing vector (red arrow) from P_c to P_m. (b) For the back-end optimization cases, when the temporal result (yellow dashed curve) of an iteration collides, we find a correspondent point P_r of the middle collision point P_c in the front-end trajectory (orange curve) and draw a pointing vector from P_r to P_c. The attracting point is chosen in its extending direction.

The selection of the attracting points is important since they guide the deformation of the optimized trajectory. For each collision part of the checked trajectory, we use local search methods like A* to find an obstacle-free path around it, and the attracting point for this collision part is chosen at some distance of the vector, which points from the middle collision position to the middle position of the A* path. This attracting point will only affect a nearby part of the trajectory around the collision part. Fig. 3a depicts this procedure. Since we only do regional optimization for relatively short local trajectories, the search area is usually very restricted, and grid search finishes within microseconds. In [25], the authors use a similar local search strategy to acquire rough distance gradient information, which can harm the convergence if possibly inaccurate. Our formulation, however, avoids this by deducing a closed-form solution and requires no gradient. The objective in our problem changes in each iteration instead. A visualization of the iterative optimization process is shown in Fig. 2.

E. Trajectory Refinement

From the kinodynamic front-end, we have an initial trajectory that settles in an appropriate homotopy class and satisfies all the constraints with the time duration reasonably allocated for each segment. However, it may not be smooth enough because of insufficient sampling within a limited time budget. As many other works [14]–[17] have validated, a lightweight refinement can greatly improve the result.

In our case, we use the same optimization framework (Alg. 3) developed for the regional optimizer, except that the **Split()** is no longer needed since the refined trajectory is already of multi-pieces. The costs are of the same formulation as Eq. 4, 5, 6, and 7. One key difference is that the search for a collision-free path is no longer needed when selecting attracting points. This is because the initial trajectory is now collision-free already. When the resulting trajectory of one iteration collides with obstacles, a point in the extending line of the vector which points from the middle collision point to the corresponding point in the initial collision-free trajectory is chosen as the attracting point for this collision part, as illustrated in Fig. 3b. By addressing the corresponding point in the initial trajectory, we mean the position at the same time stamp when the collision occurs. This correspondence is based on the added *Resemblance Cost* term and the fact that we use the front-end time allocation to initialize the back-end. This engenders similarities between the initial trajectory and the optimized trajectory. By fully exploiting the kinodynamic front-end assets and taking obstacle clearance into account while still retaining a closed-form solution for each iteration, this refinement is extremely fast and remarkably improves the success rate compared to our previous work [8].

IV. Benchmark and Experiment

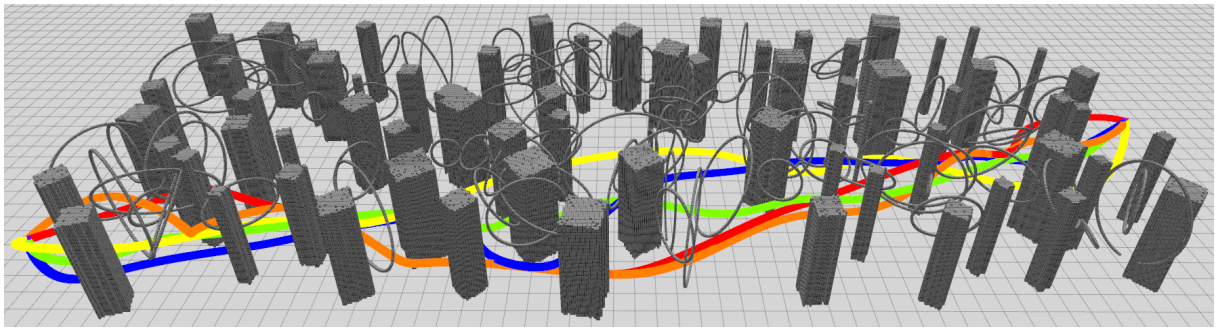
A. Experiment Settings

We set the state and control limitations of our multirotor as $5m/s$ for velocity, $7m/s^2$ for acceleration, and $15m/s^3$ for jerk. The weight of time ρ is set 100. All simulated benchmark computations are done with a 3.4GHz Intel i7-6700 processor. All real-world flights are conducted by a customized quadrotor with an onboard computer of 1.8GHz Intel i7-8550U CPU. For collision checking, the maps are pre-built and transformed into occupancy grids of $0.1m$ resolution with obstacles inflated by $0.2m$.

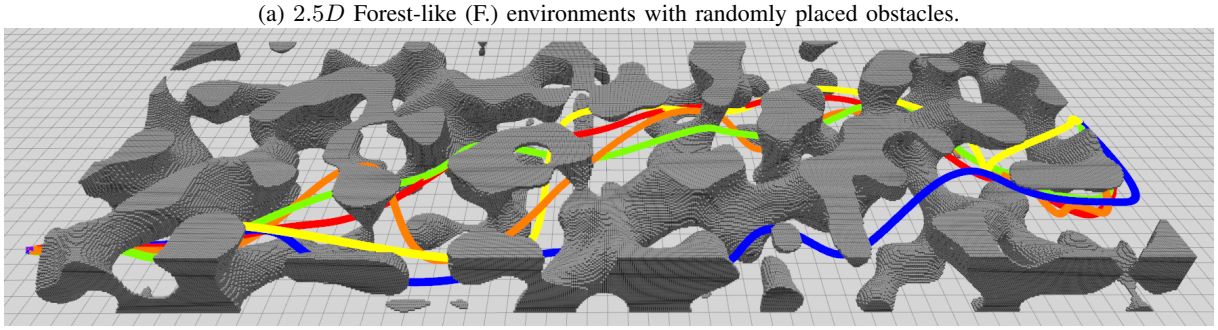
B. Front-end Comparison

To validate the performance of the proposed front-end, we carry out an ablation study of four algorithms, and the notations of which are listed in Tab.I. Each method runs for 1000 trials with different starts and goals for long-range planning in three kinds of environments, as depicted in Fig. 4 with the resulting trajectories of one trial. The same sampling strategy [8] is adopted for all the methods.

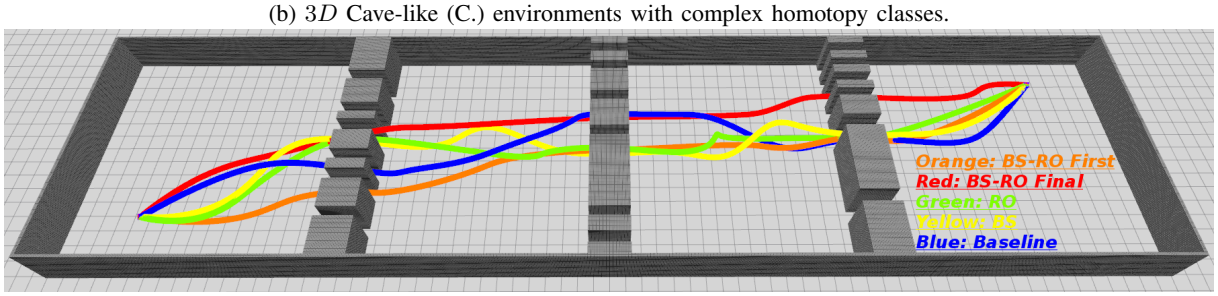
From Fig. 5, we can see that compared to the baseline, the average planning time to find the first solution is significantly



(a) $2.5D$ Forest-like (F.) environments with randomly placed obstacles.

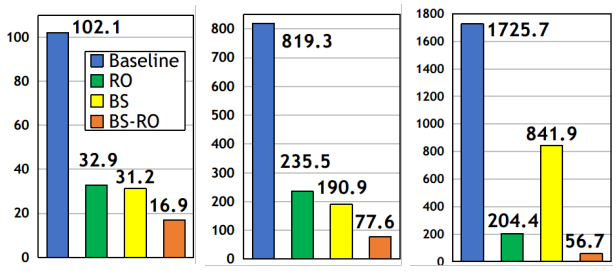


(b) $3D$ Cave-like (C.) environments with complex homotopy classes.



(c) $2.5D$ Corridor-like (W.) environments blocked by 3 thick walls with only narrow passages to travel through.

Fig. 4: Front-end trajectories in different environments. The color indications are orange for BS-RO's first trajectory, red for BS-RO's final trajectory, yellow for BS, green for RO, and blue for Baseline.



(a) Forest Env. (b) Cave Env. (c) Corridor Env.

Fig. 5: Average planning time (ms) to find the first solution in different environments.

TABLE I: Front-end methods notation.

BS-RO	kRRT* with both bidirectional search and regional optimization
BS	kRRT* with only bidirectional search
RO	kRRT* with only regional optimization
Baseline [7, 8]	standalone kRRT*

TABLE II: Front-end comparison results. Averaged over 1000 trials.

Method	Cost / 100			Success Rate (%)		
	F.	C.	W.	F.	C.	W.
BS-RO	21.6	**26.1**	**20.4**	**100.0**	**100.0**	**100.0**
BS	**21.4**	26.7	20.5	**100.0**	**100.0**	97.1
RO	22.4	26.5	21.5	**100.0**	**100.0**	**100.0**
Baseline	22.4	26.9	21.8	93.3	84.7	87.8

shortened, especially for the corridor cases with only narrow passages to travel through. Integrating the RO and the bidirectional search (BS) both contribute. Tab. II lists the average solution quality and success rate of each method, given the same planning time budget of $0.15s, 1.0s$, and $2.5s$ for F. C., and W. environment, respectively. As can be seen, integrating BS or RO or both all obtain a trajectory with better quality after the same planning time compared to the baseline. The success rate is much increased in all the testing cases. Overall the proposed method can return a feasible solution quickly within tens of milliseconds, and the solution improves with an extra time budget and thus is applicable for real-time usages.

TABLE III: Back-end comparison results. Averaged over 300 trials.

Method	Planning Time (ms)			Iteration Times (1)			Trajectory Length (m)			Jerk Integration (m^2/s^5)			Success Rate (%)		
	F.	C.	W.	F.	C.	W.	F.	C.	W.	F.	C.	W.	F.	C.	W.
Proposed	**0.27**	**0.84**	0.79	**2.0**	5.1	7.9	**64.1**	**71.7**	**66.8**	**132.1**	**187.9**	**171.5**	**97.33**	**94.00**	89.33
Base 1	0.47	1.16	**0.69**	8.1	10.9	10.9	64.3	72.9	67.4	215.3	584.8	437.2	94.67	76.67	55.33
Base 2	0.58	1.71	1.04	2.9	**3.2**	**3.5**	64.9	72.9	67.8	171.5	331.8	237.4	97.00	88.33	**95.67**
Base 3	12.1	20.1	13.0	8.2	8.1	8.9	65.0	72.4	66.9	573.7	721.6	930.4	62.33	92.67	21.33

C. Back-end Comparison

To evaluate the manners of investigating the front-end results, benchmarks are conducted with 3 other back-end methods. They are 1) our previous work (Base 1) [8] that formulates a *Homotopy Cost* to force the optimized trajectory to stay in nearby free spaces but considers no surrounding obstacle information, 2) Liu's method (Base 2) [16, 18] that only utilizes fixed intermediate waypoints and time allocation from the front-end, and 3) Zhou's method (Base 3) [15] that samples control points from the front-end trajectory and reparameterizes the result as a B-spline, and requires a distance field computed a prior (about $150ms$ in the test cases) to push the trajectory into collision-free areas. By definition, Base 1, Base 2, and the proposed method all have a closed-form solution in each iteration, whereas Base 3 formulates a soft-constrained nonlinear optimization problem that uses NLopt [26] to solve. The test environments are the same as those used in the previous section but with different random seeds. We run 300 trials for each test case, and the same front-end result of BS-RO is used for each trial.

As shown in Tab. III, the proposed back-end dominates in almost all the criteria except that in the corridor environments, the success rate is a bit lower than Base 2's, and it runs more iterations. This is because in extremely confined environments like long narrow passages that just fits the multirotor, adding fixed waypoints from the front-end trajectory in each iteration like Base 2 does can provide stronger constraints to force the optimized trajectory to stay in the free narrow passages. Note the success rate in Tab. III only limits to the back-end. For the complete planning framework, if the back-end refinement is failed, the front-end trajectory which satisfies all the constraints is used for execution.

Compared to others, the overall high performance of the proposed method can be explained by several ideas that 1) the incorporating of nearby collisions as *Collision Cost* provides richer information of the environments, and deforms the parts of the trajectory where it is originally near obstacles into obstacle-free areas, resulting in much higher success rate compared to Base 1, and 2) no requirement for any fixed waypoints provides more freedom of deformation, obtaining smoother trajectories with much lower jerk integration. Base 3, representing for those methods that use gradient descent, however requires much more time for the nonlinear programming to converge and fails a lot if no accurate distance field is available, which is usually the case in cluttered and confined environments. Under circumstances where a good front-end trajectory can be acquired, the proposed back-end can act as a great complement, providing much smoother trajectories with little effort.

D. Real-world Flight

We conduct real-world experiments in two challenging environments. In the first case, as shown in Fig. 6, the environment is maze-like with dead ends, and the start and goal positions are both trapped. The proposed methods benefit the exploration and result in better trajectories given the same planning time budget. In the second case, we further test the real-time planning capability. As shown in Fig. 7, many path homotopies exist in the cluttered environment, and there are only two circle holes to travel through in the middle. The quadrotor flies continuously in the environment and meanwhile a user randomly picks goals for it to pursue during flights (replan with initial velocity and acceleration). The proposed methods can generate smooth trajectories lying under more reasonable homotopies.

V. Conclusion

We present a kinodynamic planning method and framework that meet real-time requirements for multirotor flight. The front-end globally reasons for a trajectory by the modified kRRT* integrating bidirectional search, which explores the solution space more quickly, and an efficient and effective regional optimizer, which better handles difficult-to-sample cases. The proposed regional optimizer is reformed as the back-end refiner, which better exploits the front-end result and greatly improves it with nearly negligible computation effort. Benchmark results show the superiority of the proposed methods against other state-of-the-art multirotor kinodynamic planning methods.

References

[1] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[2] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in se(3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[3] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," *RSS: Robotics, Science, and Systems*, 2020.

[4] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *J. ACM*, vol. 40, no. 5, p. 1048–1066, Nov. 1993. [Online]. Available: https://doi.org/10.1145/174147.174150

[5] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: The drift case with linear affine dynamics," in *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 2574–2581.

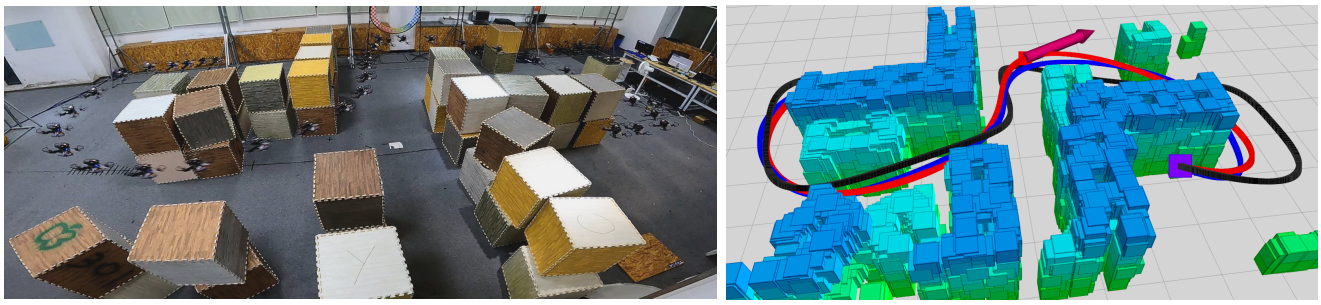


Fig. 6: Flying in a narrow maze-like environment where the start and goal positions (Purple blocks) are both trapped. Given same time budget of $100ms$, the proposed methods generate smooth trajectories (Blue for the front-end and red for the back-end), whereas the trajectory (Black) generated by the compared baseline method is more jerky and has a cost of about 15.3% higher.

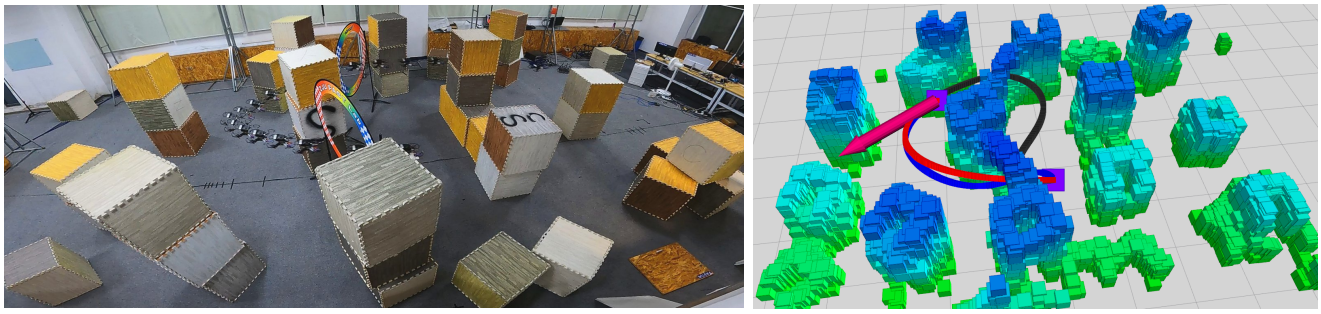


Fig. 7: Left: A composite image of random flights in a confined environment with many path homotopies. The goals are randomly picked during flights. Right: An instance showing that when replanning with an initial velocity (Magenta arrow), the proposed methods generate smooth trajectories lying under a reasonable homotopy, whereas the baseline method makes an unexpected turn.

[6] ——, "Optimal sampling-based motion planning under differential constraints: The driftless case," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2368–2375.

[7] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, May 2013, pp. 5054–5061.

[8] H. Ye, X. Zhou, Z. Wang, C. Xu, J. Chu, and F. Gao, "Tgk-planner: An efficient topology guided kinodynamic planner for autonomous quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 494–501, 2021.

[9] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in rrt* for optimal kinodynamic planning," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 3737–3744.

[10] A. Kuntz, C. Bowen, and R. Alterovitz, "Interleaving optimization with sampling-based motion planning (IOS-MP): combining local optimization with global exploration," 2016. [Online]. Available: http://arxiv.org/abs/1607.06374

[11] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, "Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4207–4214.

[12] D. Kim, Y. Kwon, and S. Yoon, "Dancing prm*: Simultaneous planning of sampling and optimization with configuration free space approximation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7071–7078.

[13] D. Kim, M. Kang, and S. Yoon, "Volumetric tree*: Adaptive sparse graph for effective exploration of homotopy classes," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1496–1503.

[14] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[15] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[16] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, Sept 2017, pp. 2872–2879.

[17] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 344–351.

[18] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *Intl. J. Robot. Research (IJRR)*, vol. 34, no. 7, pp. 969–1002, 2015.

[19] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[20] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3067–3074.

[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846–894, 2011.

[22] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, Shanghai, China, May 2011, pp. 2520–2525.

[23] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters (RA-L)*, pp. 1688–1695, 2017.

[24] Z. Wang, H. Ye, C. Xu, and F. Gao, "Generating large-scale trajectories efficiently using double descriptions of polynomials," *arXiv preprint arXiv:2011.02662*, 2020.

[25] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2021.

[26] S. G. Johnson, "The nlopt nonlinear-optimization package." [Online]. Available: http://github.com/stevengj/nlopt