2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Search-Based Online Trajectory Planning for Car-like Robots in Highly Dynamic Environments

Jiahui Lin[1], Tong Zhou[1], Delong Zhu[1*], Jianbang Liu[1], and Max Q.-H. Meng[2*]

***Abstract*— This paper presents a search-based partial motion planner for generating feasible trajectories of car-like robots in highly dynamic environments. The planner searches for smooth, safe, and near-time-optimal trajectories by exploring a state graph built on motion primitives. To enable fast online planning, we propose an efficient path searching algorithm based on the aggregation and pruning of motion primitives. We then propose a fast collision checking algorithm that takes into account the motions of moving obstacles. The algorithm linearizes relative motions between the robot and obstacles, and then checks collisions by calculating a point-line distance. Benefiting from the fast searching and collision checking algorithms, the planner can effectively explore the state-time space to generate near-time-optimal solutions. Experiments show that the proposed method can generate feasible trajectories within milliseconds while maintaining a higher success rate than up-to-date methods, which significantly demonstrates its advantages.**

I. Introduction

Online trajectory planning in dynamic environments has a wide application in autonomous robotic systems [1]–[4]. However, it is a challenging task due to the unpredictable future motions of moving obstacles. The nonholonomic model and time-dependent collision checking impose nonlinear constraints on the planning task, making it even more intractable. The optimal solution is typically unachievable, and the completeness cannot be ensured [5]. Existing studies mostly focus on finding near-optimal partial solutions within a specific planning horizon to approximate the global solution with the help of a fast re-planning process.

Partial Motion Planning (PMP) in dynamic environments is first proposed by Petti *et al.* [6] within the framework of Rapidly Exploring Random Tree (RRT). Since in dynamic environments, the randomized sampling approach, e.g., RRT, takes a long time to converge, the authors allow a time-bounded partial solution with its terminal state ICS-free [7]. In this way, the planning and execution procedures can be accurately scheduled to achieve fast re-planning. Based on this idea, more sampling-based methods, e.g., [8], are proposed to improve the optimality of partial trajectories. The idea of PMP is also adopted by searching-based algorithms [9]–[12], which combine a long-horizon path searcher to improve the global optimality. The major deficiency of these methods lies in planning inefficiency. There typically exist a large number of candidate paths in a time-involved planning problem, and the collision checking at each searching step also needs a complicated procedure to cope with obstacle motions, which significantly limits the planning speed. Meanwhile, the differential constraints reduce the volume of admissible state space, making the problem even more difficult.

To address these problems, we propose an online partial motion planner based on motion primitives. We first discretize the time dimension and control space of the robot to generate motion primitives, which are short trajectories within a specified duration. Continuous expansion of the primitives generates a primitive tree that covers the state space. Direct searching on the primitive tree is a time-consuming process. We thus also discretize the state space into grids. Primitives that lie in the same grid are aggregated and then pruned. The pruning operation is carefully designed, which can significantly reduce the searching scale while maintaining the smoothness at pruning points. Furthermore, we propose an efficient collision checking algorithm that takes into account the motions of obstacles (modeled by velocity extrapolation [5]). The algorithm performs velocity planning on the primitives and can provide a strict safety guarantee under the velocity extrapolation model.

The contributions of this work are as follows:

- We propose an online graph searching algorithm based on motion primitives. By leveraging node aggregation and pruning, the algorithm can efficiently explore the state space and generate near-time-optimal solutions.
- We propose a fast collision checking algorithm based on the linearization of relative motions between the robot and moving obstacles, which significantly improves the safety of the generated trajectory.
- Based on the above two algorithms, we implement an online trajectory planning framework with very high planning frequency and success rates.

II. Related work

Motion primitive is a frequently used method by the planning community. LaValle *et al.* [13] propose a node expansion algorithm using motion primitives to enable kinodynamic planning with RRT. The idea is to approach the target state with motion primitives to make the trajectory

* The corresponding author of this paper.
[1] The authors are with the Department of Electronic Engineering, The Chinese University of Hong Kong, China. *email: zhudelong@link.cuhk.edu.hk, jiahuilin@cuhk.edu.hk, tzhou@ee.cuhk.edu.hk.*
[2] Max Q.-H. Meng is with the Department of Electronic and Electrical Engineering, Southern University of Science and Technology, Shenzhen, China, and also with the Shenzhen Research Institute of the Chinese University of Hong Kong, Shenzhen, China, on leave from the Department of Electronic Engineering of the Chinese University of Hong Kong, China (e-mail: max.meng@ieee.org). This project is partially supported by the Hong Kong RGC GRF grants #14200618 and Hong Kong ITC ITSP Tier 2 grant #ITS/105/18FP awarded to Max Q.-H. Meng.

satisfy kinodynamic constraints. However, the primitives used in [13] are generated by sampling the control space rather than solving the two-point Boundary Value Problem (BVP), hence the continuity of the trajectory is not ensured. This problem is then solved by Palmieri *et al.* [14] with an efficient BVP solver. Liu *et al.* [15] combine the graph searching method with motion primitives to approximate the optimal control problem for quadrotors, achieving aggressive flight in SE(3) space [16]. The above methods are designed for static environments but share a similar idea with ours.

State lattice introduced by [17] for planning in static environments, is another discretization method that transforms the continuous state space into searching graphs.The edges that connect adjacent states in state lattice are another type of motion primitive. McNaughton *et al.* [10] extend the state lattice with a time dimension, which allows the search algorithm to explore both spatial and temporal dimensions efficiently. Kushleyev *et al.* [11] propose a time-bounded lattice for planning in dynamic environments. The A* is used to search on a pre-computed lattice. However, the collision checking in this work is conducted on the discretized waypoints, thus cannot provide a safety guarantee.

Trajectory library is the third type of motion primitive. The idea is to build a prior road map that respects motion constraints for the target environment, and then select the optimal path with online collision checking. Zhang *et al.* [18] adopt the BIT* method [19] to pre-build a prior map for the environment, based on which an online collision checking process is conducted to select the best trajectory. In this way, the computation cost is significantly reduced. Some other studies that make use of prior maps include Vector Field based method [20] and Voronoi Random Field based method [21]. These methods aim to accelerate the online processing by downloading part of the computation to an offline process. The trajectory library based method is capable of dealing with dynamic environments to some extent. For highly dynamic environments, the number of trajectories that need collision checking would be intractable.

III. Problem Definition

The planning problem in this work is defined as a tuple $< \mathcal{R}, \mathcal{M}, \mathcal{B}, \mathcal{J} >$, which denotes the motion model, the map, the boundary value, and the planning objective.

The motion model $\mathcal{R}$ is defined by its state function with differential constraints,

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}), \\ h(\mathbf{x}, \mathbf{u}) &= 0,\ h(\mathbf{x}, \mathbf{u}) \leq 0, \end{aligned} \tag{1}$$

where $\mathbf{x}$ and $\mathbf{u}$ are robot state and control input, respectively.

The map $\mathcal{M}$ defines a model for the environment, which separates the free space $\mathcal{M}_f$ from static obstacles. In dynamic environments, due to the presence of moving obstacles, $\mathcal{O}_i(t), i \in \{1, \cdots, M\}$, the free space is further restricted, which imposes safety constraints,

$$\mathbf{x}(t) \notin \mathcal{M}_f \cap \mathcal{O}_i(t),\ \forall i \in \{1, \ldots, M\}, \forall t \in [t_0, t_0 + T], \tag{2}$$

where t_0 is the start time of collision checking, and T is the planning horizon. One of the key challenges of planning in dynamic environments is the calculation of such constraints.

The boundary value $\mathcal{B}$ specifies a planning start $\mathbf{x}_0$, a goal set $\mathcal{X}_g$, and an intermediate point $\mathbf{x}_T \notin \mathcal{X}_{ICS}$. For planning without moving obstacles, the intermediate point is not necessary, since any point in a successfully planned path is ensured to be collision-free during execution. However, this condition does not hold when there exist moving obstacles, since their motions are unpredictable. We thus need to specify a local ICS-free goal to ensure the robot can safely navigate there before the next trajectory is available.

$$\mathbf{x}(t_0) = \mathbf{x}_0,\ \mathbf{x}(t_0 + T_g) \in \mathcal{X}_g,\ \mathbf{x}(t_0 + T) \notin \mathcal{X}_{ICS}. \tag{3}$$

The planning objective in this work is to generate a trajectory that is smooth, collision-free, respects the motion constraints, and has minimum execution time. The smoothness is defined as the squared L2-norm of the control efforts,

$$J = \int_0^{T_g} \|\mathbf{u}(t)\|^2 dt. \tag{4}$$

Based on the above definition, we formulate the dynamical motion planning into an optimization problem,

$$\begin{aligned} &\min_{\mathbf{x}(t), \mathbf{u}(t), T_g} J + \beta T_g \\ &\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}),\ h(\mathbf{x}, \mathbf{u}) = 0,\ h(\mathbf{x}, \mathbf{u}) \leq 0, \\ &\mathbf{x}(t) \notin \mathcal{M}_f \cap \mathcal{O}(t),\ t \in [t_0, t_0 + T], \\ &\mathbf{x}(t_0) = \mathbf{x}_0, \mathbf{x}(t_0 + T_g) \in \mathcal{X}_g,\ \mathbf{x}(t_0 + T) \notin \mathcal{X}_{ICS}. \end{aligned} \tag{5}$$

The solution for this problem defines a partial trajectory within the planning horizon T. For the remaining part in $(T, T_g]$, the safety constraint is not required to be strictly satisfied. This is different from the conventional motion planning and is referred to as the PMP. We approximate the solution with motion primitives at the expense of optimality.

IV. Graph Searching with Motion Primitives

A. Robot Motion Model

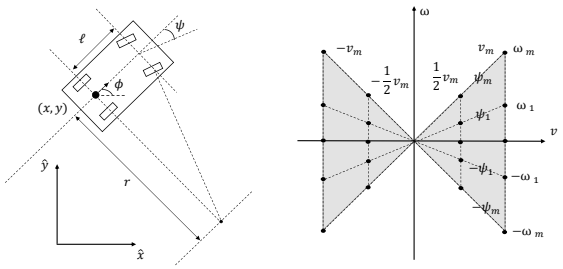
In this work, as shown in Fig. 1a, we use the canonical simplified car kinematics to model the robot,

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{bmatrix} = B(\mathbf{x})\mathbf{u} = \begin{bmatrix} 0 & 1 \\ \cos\phi & 0 \\ \sin\phi & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{6}$$

$$v = v_0 + a * t,\ \omega = (\tan\psi)/\ell * v, \tag{7}$$

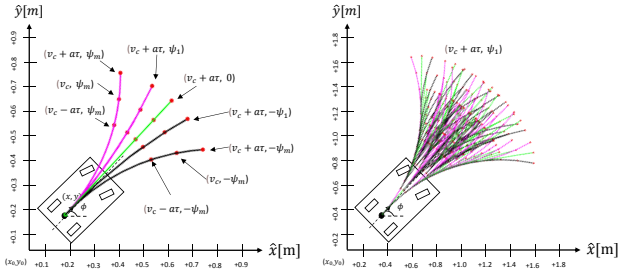
where $\mathbf{x}$=$(x, y, \phi) \in \mathcal{X}$ defines the robot state, which includes the heading direction ϕ and robot location (x, y), and ψ is the steering angle. The control input (v, ω) includes the forward speed and rotation rate of the robot. A feasible control set is visualized in Fig. 1b, where v and ω are linearly related, as indicated by Eq. (7). The slope rate of the lines in the bowtie control set is determined by the maximum steering angel ψ (or the minimum turning radius) of the robot.

In practice, more frequently-used control input is the steering angle and linear acceleration, i.e., $\mathbf{u} = (\psi, a)$. Each ψ corresponds to a line in Fig. 1b with its slope rate defined by $(\tan\psi)/\ell$. If we denote the current velocity of the robot by v_c and the steering angle is not changed, the subsequent velocities accelerated by a will fall on this line. To make the generated trajectory dynamically feasible, we should ensure the control inputs always lie in the control set, i.e., $\mathbf{u}(t) \in \mathcal{U}$.



(a) The simplified car kinematics. (b) The (v, w) control set.

Fig. 1: The motion and control models for the wheeled robot.



(a) One-layer motion primitives. (b) Motion primitive tree.

Fig. 2: The generation of motion primitives based on a car model.

B. Motion Primitives

Motion primitive $\boldsymbol{\xi}_k$ is a short trajectory within duration τ, which can bring the robot from the current state $\mathbf{x}_k$ to a new state $\mathbf{x}_{k+1}$ while respecting the motion constraints. Denote by $\mathcal{U}_N = \{\mathbf{u}_1, \dots, \mathbf{u}_N\} \subset \mathcal{U}$ the discretized control inputs, the motion primitives for $t \in [0, \tau]$ that apply $\mathbf{u}_n \in \mathcal{U}_N$ are

$$\begin{aligned} \phi(t) &= \phi_k + (v_k t + \frac{1}{2} a_n t^2)(\tan\psi_n)/\ell, \\ x(t) &= x_k + \sin(\phi(t))\, \ell/(\tan\psi_n), \\ y(t) &= y_k - \cos(\phi(t))\, \ell/(\tan\psi_n), \end{aligned} \tag{8}$$

where $(\tan\psi_m)/\ell = 1/r$ and r is the turning radius (Fig. 1a) of the primitive. Sometimes v_k may reach its maximum before τ is used up. In this case, the remaining of the motion primitive is generated with the maximum velocity.

We visualize the one-layer motion primitives starting at $\mathbf{x}_k$ in Fig. 2a. Each arch is specified by a sampled steering angel, and each red point in the arch is a terminal state of the motion primitive generated with a sampled linear acceleration. Starting from the current state, we can forward sample the control inputs and continuously grow the motion primitives until the considered state space is sufficiently expanded. We demonstrate such an expansion process in Fig. 2b, which is essentially a tree and thus named primitive tree.

C. Online Graph Searching

Motion primitive $\boldsymbol{\xi}_k$ is uniquely determined by a tuple $(\mathbf{x}_k, \mathbf{u}_n, \tau)$, which is a sample of the solution space in Eq. (5). The cost of each primitive is $\|\mathbf{u}(t)\|^2 + \tau$, where $\|\mathbf{u}(t)\|^2 = p_1 a^2 + p_2 \psi^2$. By leveraging motion primitives, the PMP defined in Eq. (5) is transformed into a combinatorial

Algorithm 1 GeneratePrimitives($\mathbf{s}_k$, $\mathcal{X}_g$, $\mathcal{O}$)

```
1:  SUCC ← ∅
2:  for u ∈ U_N do
3:      ξ_k, s_{k+1} ← MOTIONPRIMITIVE(s_k, u, τ)
4:      if ISVALID(s_{k+1}) ∧ COLLIFREE(ξ_k, O, τ) then
5:          g(s_{k+1}) ← g(s_k) + PIRITIVECOST(ξ_k)
6:          h(s_{k+1}) ← HEURISTIC(s_{k+1}, X_g)
7:          f(s_{k+1}) ← g(s_{k+1}) + h(s_{k+1})
8:          s_{k+1}.pre ← s_k
9:          INSERT(s_{k+1}, SUCC)
10:     end if
11: end for
12: return SUCC
```

optimization problem w.r.t. $\boldsymbol{\xi}_{0:K}$ and K,

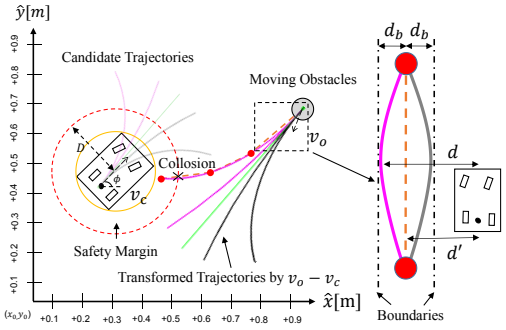
$$\begin{gathered} \min_{\boldsymbol{\xi}_{0:K},\, K} \sum_{k=0}^{K} \left(\|\mathbf{u}_k\|^2 + \tau\right) \\ \boldsymbol{\xi}_k((k+1)\tau) = \boldsymbol{\xi}_{k+1}((k+1)\tau),\ \forall \boldsymbol{\xi}_i \in \Xi_i, \\ \boldsymbol{\xi}_k(t) \notin \mathcal{M}_f \cap \mathcal{O}(t),\ t \in [k\tau, (k+1)\tau], \forall k \in \{0, .., \lceil T/\tau \rceil\}, \\ \boldsymbol{\xi}_0(0) = \mathbf{x}_0, \boldsymbol{\xi}_K \cap \mathcal{X}_g \neq \emptyset,\ \boldsymbol{\xi}_{\lceil T/\tau \rceil} \cap \mathcal{X}_{ICS} = \emptyset, \end{gathered} \tag{9}$$

where Ξ_i is the set of primitives in $[i\tau, (i+1)\tau]$ with a depth of i in the tree. This problem is typically solved by tree searching, e.g., BFS, which however is not suitable for online trajectory generation due to its inefficiency. Here, we propose to additionally discretize the state space for aggregating motion primitives to enable fast trajectory generation.

In Alg. 1, we present the generation of collision-free motion primitives for growing the primitive tree (see Section IV-D). The primitive cost is defined in Eq. (9). The heuristic cost is defined as the length of Reed-Shepp curves without considering moving obstacles. In Alg. 2, we perform graph building and path searching using a variant of A* algorithm. The HMAP is a hash map for holding the discretized state graph, which is a grid discretization of $(x, y, \phi) \in \mathcal{X}$. Each state node is induced by a primitive and belongs to one of the grid cells. We use a volumetric hash key $\mathbf{s}.\mathrm{key}$ to connect the node and the state grid. The state grid is used to aggregates motion primitive for pruning. In lines 7-14, we check whether the goal or planning horizon is achieved, and make sure the local goal $\mathbf{s}_t$ is ICS-free. Lines 16-35 present the node expansion and trajectory generation process. If a grid cell is never reached before, we just generate a new state node for this cell (lines 32-34); Otherwise, we need to update the optimal representation for this cell (lines 22-29). In Fig. 4, we visualize three cases (lines 26-28) that require to update the optimal representation, which is also the key implementation of aggregating motion primitives and reducing search efforts.

D. Collision Checking

Collision checking is one of the most time-consuming processes in dynamic motion planning. A typical method is to discretize the target trajectory and then check collisions for each discretized waypoint by a forward simulation process.



(a) Illustration of the linearized collision checking.

(b) Illustration of the process of ICS checking.

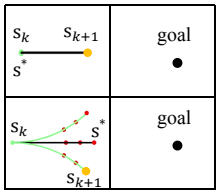
Fig. 3: Collision and ICS checking during graph searching.

Algorithm 2 GraphSearching($\mathbf{s}_0$, $\mathcal{X}_g$, $\mathcal{O}$, τ, T)

HMAP $\leftarrow \emptyset$, OPEN $\leftarrow \emptyset$
$\mathbf{s}_0$.key $\leftarrow$ GenKey, ($\mathbf{s}_0$) g($\mathbf{s}_0$) $\leftarrow 0$
HMAP.Insert($\mathbf{s}_0$), OPEN.Insert($\mathbf{s}_0$)
while OPEN is not empty **do**
  $\mathbf{s}_k \leftarrow$ OPEN.Pop()
  $\mathbf{s}_k$.closed $\leftarrow$ true
  $\mathbf{s}_g \leftarrow$ GoalReached($\mathbf{s}, \mathcal{X}_g$)
  $\mathbf{s}_t \leftarrow$ TimeReached($\mathbf{s}, T$)
  /* *the goal or planning horizon is reached* */
  if $\mathbf{s}_g$.state $\in \mathcal{X}_g$ **then**
    return BackTrack($\mathbf{s}_g$)
  else if $\mathbf{s}_t$.state $\notin \mathcal{X}_{ICS}$ **then**
    return BackTrack($\mathbf{s}_t$)
  end if
  /* *generate successors using motion primitives* */
  Succ $\leftarrow$ GeneratePrimitives($\mathbf{s}_k$, $\mathcal{X}_g$, $\mathcal{O}$)
  for $\mathbf{s}_{k+1} \in$ Succ **do**
    $\mathbf{s}_{k+1}$.key $\leftarrow$ GenKey($\mathbf{s}_{k+1}$)
    elements $\leftarrow$ HMAP.Query($\mathbf{s}_{k+1}$.key)
    /* *already reached by other primitives* */
    if elements $\neq \emptyset$ **then**
      $\mathbf{s}^* \leftarrow$ elements.best
      case1 $\leftarrow \mathbf{s}^*$.key $= \mathbf{s}$.key $\vee$ $\mathbf{s}^*$.pre $= \mathbf{s}_{k+1}$.pre
      case2 $\leftarrow \sim$case1
      if case1 $\wedge$ (h($\mathbf{s}_{k+1}$) $<$ h($\mathbf{s}^*$)) **then**
        HMAP.Insert($\mathbf{s}_{k+1}$), OPEN.Insert($\mathbf{s}_{k+1}$)
      end if
      if case2 $\wedge \sim \mathbf{s}^*$.closed $\wedge$ g($\mathbf{s}_{k+1}$) $<$ g($\mathbf{s}^*$) **then**
        HMAP.Replace($\mathbf{s}^*, \mathbf{s}_{k+1}$)
      end if
    /* *never reached by other primitives* */
    else
      HMAP.Insert($\mathbf{s}_{k+1}$), OPEN.Insert($\mathbf{s}_{k+1}$)
    end if
  end for
end while
return Failure

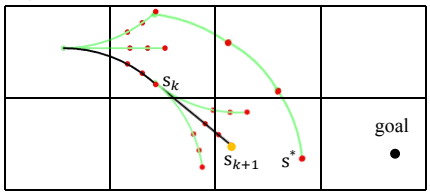
Such a method cannot ensure the safety between waypoints, which may lead to severe collisions when there exist high-speed obstacles. To address this problem, we propose to calculate the minimum distance d between motion primitive $\boldsymbol{\xi}_k$ and the obstacle trajectory for $t \in [k\tau, (k+1)\tau]$. If the condition $d > D$ is satisfied, where D is the safety margin, the safety of $\boldsymbol{\xi}_k$ will be ensured.

To achieve this point, as shown in Fig. 3a, we first transform the obstacle trajectory by subtracting $\boldsymbol{\xi}_k$, which can be easily implemented by using Eq. (8). Based on such a transformation, the problem is then converted to checking the minimum distance d between the current position of the robot and the transformed trajectory of the obstacle. To accelerate the collision checking process, we only generate several waypoints (the red points in Fig. 3a) and then connect these waypoints as line segments to linearize the transformed trajectory. As shown in the right side of Fig. 3a, after the linearization, d is then bounded within $[d' - d_b, d' + d_b]$, where d' is the minimum distance between the robot and the line segment. Therefore, the safety constraint in Eq. (9) can be satisfied, if $d' - d_b > D$ is ensured. Through the trajectory transformation and linearization, the collision checking is completed sufficiently and efficiently.



(a) case1: $\mathbf{s}_k$ and $\mathbf{s}_{k+1}$ are in the same grid cell, and $\mathbf{s}^*$ is the parent or sibling of $\mathbf{s}_{k+1}$.

(b) case2: $\mathbf{s}_k$ and $\mathbf{s}_{k+1}$ connect two grid cells, and $\mathbf{s}^*$ is the sibling of $\mathbf{s}_{k+1}$.



(c) case3: $\mathbf{s}_k$ and $\mathbf{s}_{k+1}$ connect two grid cells, and $\mathbf{s}^*$ is not the parent or sibling of $\mathbf{s}_{k+1}$.

Fig. 4: Illustration of the three cases of node expansion. $\mathbf{s}^*$ is the optimal representation of the current grid cell. $\mathbf{s}_k$ is the parent node and $\mathbf{s}_{k+1}$ is one of the child nodes being evaluated. The yellow point is the newly selected optimal representation of the current grid cell.

Another safety constraint is imposed on the local goal, i.e., $\mathbf{x}_t \notin \mathcal{X}_{ICS}$. The ICS is defined as a state such that once the robot navigates into this state, it will inevitably collide with obstacles within a time interval. Identifying ICS is a time-consuming task, and thus in this work, we propose to test all the least-acceleration motion primitives starting from $\mathbf{x}_t$, demonstrated in Fig. 3b, using our collision checking method. If there exists at least one collision-free primitive, it can be ensured that $\mathbf{x}_t \notin \mathcal{X}_{ICS}$. Here, we only need to check the least-acceleration primitive for each steering angle (the most internal points in Fig. 2a), since if they are not ICS free, the outer primitives will not be ICS free.

V. Experiments

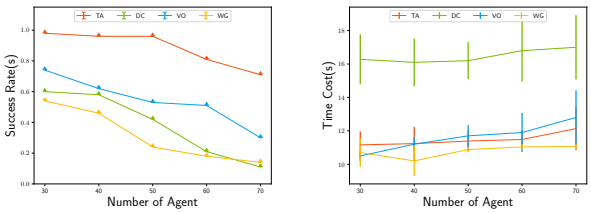
In this section, we first evaluate our method on a simulated environment to study its performance changes under different parameter settings. We then perform an overall evaluation on three benchmark datasets to compare our method with exiting work in the literature. Three baseline methods are adopted: the passive wait-and-go (WG) strategy, the classical velocity obstacle (VO) method [22], and the up-to-date dynamic channel (DC) method [23]. The experiment platform is a laptop with i5-9400 CPU 2.90 GHz with 8 GB of RAM.

A. Experiment on Simulation Environment

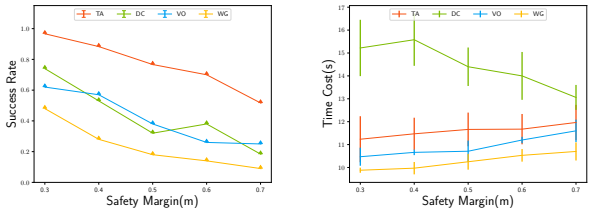
The performance of trajectory planning in dynamic environments is primarily influenced by three factors: the maximum velocity of the robot, the number of moving obstacles, and the safety margin. To quantify the influence, we conduct three control experiments in a 10m-by-10m simulation environment with multiple mobile agents. The moving direction of each agent is randomly initialized. The speed is generated by a uniform distribution from 1.2 to 2.0 m/s. If an agent moves outside the environment, a new agent will respawn. In this way, a constant obstacle density is maintained. The current motion of the agents is available to the robot but with uncertainty on speed, which is modeled with a Gaussian noise $\mathcal{N}(0, 0.1)$.

The default parameter settings are as follows: 40 moving agents, a safety margin of 0.3m, and a maximum linear speed of 1.8m/s for the robot. In each test, only one of the parameters is changed. The middle points on the left and right sides of the environment are taken as planning start and goal, respectively. The success times and time cost of 30 repeat tests are reported as evaluation metrics. During experiments, a test is counted as a failure, if a collision happens or the robot cannot reach the goal in 30 seconds. The results are shown in Fig. 5.

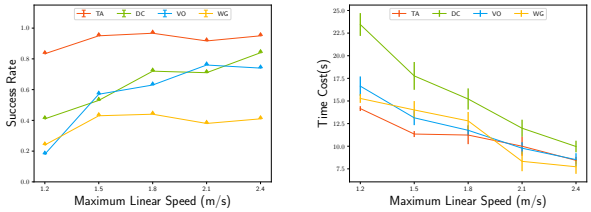
We can see that all planners exhibit a worse performance when the number of moving agents or the safe margin is increased, see Fig. 5a and 5b. The reason is obvious that the feasible region in the planning space is occupied by obstacles and margins, which makes it more difficult for the planner to find a feasible solution. Compared with the classical VO method, the latest DC planner shows a worse success rate in Fig. 5a, while maintaining a competitive performance in Fig. 5b. This is essentially caused by different collision checking



(a) Results under different number of agents.



(b) Results under different safety margins.



(c) Results under different maximum linear speeds.

Fig. 5: Performance on the simulation environment.

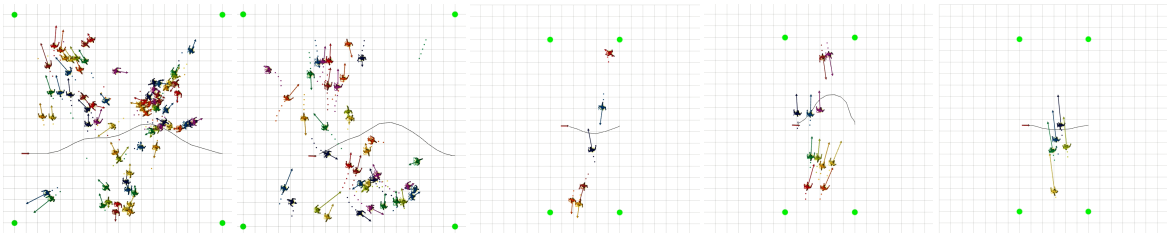
strategies. The DC method performs path searching on a sparse triangle graph, and its collision checking is limited to passing through a gate defined by the common edge of the adjacent triangles, which cannot provide a safety guarantee, especially in highly dynamic environments. The VO method avoids the surrounding obstacles aggressively, thus exhibiting a better performance when the number of moving agents is increased. The proposed TA method can be regarded as a variant of VO enhanced by long-horizon planning, and thus shows the best performance among the four planners.

In terms of the maximum linear velocity in Fig. 5c, the success rate of DC is positively related to this factor, while the other three planners show a decreasing tendency after a certain test point. For the DC method, a higher speed can enable faster navigation through the gate under smaller topology changes of the graph, hence the success rate increases. For the TA and VO method, higher speeds increase the reactive ability of the robot and thus increase the success rate. However, higher speeds increase the horizon of collision checking, which is not beneficial for fast-planning. The success rate is thus affected to some extent.

The simulation experiment shows that our proposed TA method can achieve the best performance on success rate and time cost, which significantly demonstrates its advantages.

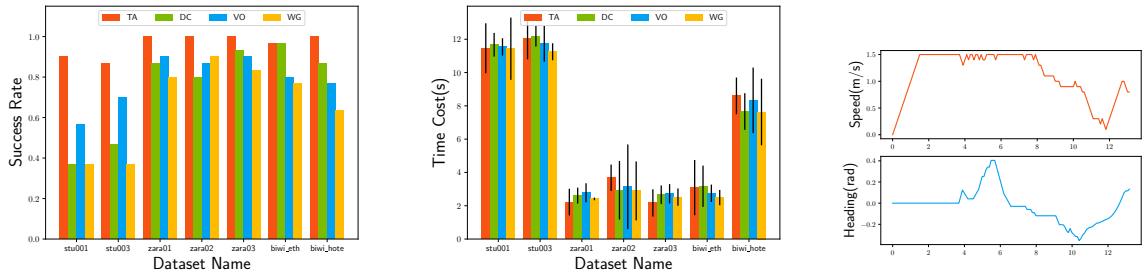
B. Experiment on Benchmark Datasets

We then evaluate our method on three public publicly available datasets from ETH [24] and UCY [25]. There are totally seven sequences of videos captured from different scenarios. All the sequences are interpolated and played at



(a) stu001 (b) stu003 (c) zara01 (d) zara03 (e) biwi_eth

Fig. 6: The planning results of our online state-time planner on different data sequences.



(a) The results of success rate. (b) The results average time cost. (c) Speed and heading of robot

Fig. 7: Performance on the benchmark datasets.

a frequency of 10Hz. As demonstrated in Fig. 6, the *stu*, *zara*, and *biwi* sequences are with a high, medium, and low crowd density, respectively. During experiments, the maximum linear speed of the robot is set to 1.5m/s. The safe distance is 0.4m. For each sequence, 30 tests are performed, and each test starts at a random time point of the sequence. The same evaluation metrics with the simulation experiment are adopted, and the results are presented in Fig. 7.

We can see WG achieves the lowest time cost in all sequences with a low success rate in *stu001* and *stu003*, see Fig. 8a and 8b. On the one hand, WG always chooses the shortest path which produces the lowest time cost. On the other hand, the waiting strategy has a large chance to collide with the crowd, which results in a low success rate. As VO avoids the obstacle aggressively without considering the future motion of the crowd, its performance on time cost is unstable. The fact that VO has time cost with large variance in *zara01*, *zara02* and *biwi_hotel* also support this statement. Time cost of TA is the highest in *zara02* and *biwi_hotel*. The reasons are as follows: Firstly, TA can find a complex path toward the goal in dense situations that produce a high time cost. As shown in Fig 8a, TA still achieves a high success rate in *zara02* and *biwi_hotel*. Secondly, TA may prefer a longer trajectory by going around the obstacle as a penalty is given to deacceleration. We still claim that TA shows advantages over other planners, because TA shows the highest success rate and maintains a relatively low time cost in all sequences. In general, we can observe a similar phenomenon in the simulation environment. As shown in Fig. 7a, both TA and VO methods achieve a higher success rate in dense sequences, while DC performs better than VO in sparse sequences.

Fig. 7c shows the state-time graph of a trajectory generated by TA planner. We can observe that the speed and heading are smooth at the beginning, as TA prefers to go straight toward the goal without deacceleration. The rapid changes in linear speed are caused by unexpected collision avoidance, as the constant velocity assumption does not hold sometimes. The computation time of TA in different benchmark datasets is shown in Table I. Some key parameters related to the computation time are as follows: 9 primitives for each step, 0.5s for each planning horizon, $\alpha = 1.3$ for scaling heuristic. We can see that the proposed planner has very high efficiency.

TABLE I: Computation Time on Benchmark Dataset

Scene	stu001	stu003	zara01	zara02	eth	hotel
mean(ms)	3.14	4.77	3.14	3.64	1.85	3.12
std(ms)	1.67	2.61	2.12	2.50	0.98	1.25

VI. Conclusions

In this work, we propose a search-based partial motion planner for generating dynamically feasible trajectories of car-like robots in highly dynamic environments. The primitives generated by discretized control is over densely distributed in state-space, which is the major deficiency of control space discretization method. We tackle this problem by selecting a set of optimal primitives for each discrete cell. We also propose a fast collision checking algorithm to further accelerate the planning process. Extensive experiments demonstrate the significant advantages of our method.

As the algorithm is based on space discretization, both the state grid and control set depend on the scale of scenarios. To balance the optimality and efficiency, we have to carefully select the discretization resolution. In the next step, we will integrate more informative heuristic and accurate motion models of the obstacles to further improve the performance of our framework.

REFERENCES

[1] D. Zhu, Y. Du, Y. Lin, H. Li, C. Wang, X. Xu, and M. Q. . Meng, "Hawkeye: Open source framework for field surveillance," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6083–6090.

[2] D. Zhu, T. Li, D. Ho, C. Wang, and M. Q. . Meng, "Deep reinforcement learning supervised autonomous exploration in office environments," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7548–7555.

[3] T. Li, J. Pan, D. Zhu, and M. Q.-H. Meng, "Learning to interrupt: A hierarchical deep reinforcement learning framework for efficient exploration," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2019.

[4] C. Wang, D. Zhu, T. Li, M. Q. . Meng, and C. W. de Silva, "Efficient autonomous robotic exploration with semantic road map in indoor environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2989–2996, July 2019.

[5] H.-T. L. Chiang, B. HomChaudhuri, L. Smith, and L. Tapia, "Safety, challenges, and performance of motion planners in dynamic environments," in *Robotics Research*. Springer, 2020, pp. 793–808.

[6] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005, pp. 2210–2215.

[7] T. Fraichard and H. Asama, "Inevitable collision states. A step towards safer robots?" *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.

[8] H. L. Chiang, B. HomChaudhuri, A. P. Vinod, M. Oishi, and L. Tapia, "Dynamic risk tolerance: Motion planning by balancing short-term and long-term stochastic dynamic predictions," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3762–3769.

[9] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," in *2007 IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 1986–1991.

[10] M. McNaughton, C. Urmson, J. M. Dolan, and J. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4889–4895.

[11] A. Kushleyev and M. Likhachev, "Time-bounded lattice for efficient planning in dynamic environments," in *2009 IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 1662–1668.

[12] M. Rufli and R. Siegwart, "On the application of the d* search algorithm to time-based planning on lattice graphs," in *European Conference on Mobile Robots*, 2009.

[13] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *1999 IEEE International Conference on Robotics and Automation (ICRA)*, 1999, pp. 473–479.

[14] L. Palmieri and K. O. Arras, "A novel rrt extend function for efficient and smooth mobile robot motion planning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 205–211.

[15] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2872–2879.

[16] 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.

[17] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," *International Symposium on Artificial Intelligence Robotics and Automation in Space*, vol. 603, no. 603, 2008.

[18] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-cal: Precomputed alternative lanes for aggressive aerial collision avoidance," in *12th Conference on Field and Service Robotics*, 2019.

[19] 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.

[20] G. A. S. Pereira, S. Choudhury, and S. Scherer, "A framework for optimal repairing of vector field-based motion plans," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016, pp. 261–266.

[21] P. Beeson, N. K. Jong, and B. Kuipers, "Towards autonomous topological place detection using the extended voronoi graph," in *2005 IEEE International Conference on Robotics and Automation (ICRA)*, 2005, pp. 4373–4379.

[22] D. Wilkie, J. van den Berg, and D. Manocha, "Generalized velocity obstacles," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009, pp. 5573–5578.

[23] C. Cao, P. Trautman, and S. Iba, "Dynamic channel: A planning framework for crowd navigation," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 5551–5557.

[24] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 261–268.

[25] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," *Computer Graphics Forum*, vol. 26, no. 3, pp. 655–664, 2007.