

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

Jiahui Lin¹, Tong Zhou¹, Delong Zhu^{1*}, Jianbang Liu¹, and Max Q.-H. Meng^{1,2*}

Abstract—This paper presents a search-based partial motion planner to generate dynamically feasible trajectories for 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, which are generated by discretizing the time dimension and the control space. To enable fast online planning, we first 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 comparing a point-line distance. Benefiting from the fast searching and collision checking algorithms, the planner can effectively and safely explore the state-time space to generate near-time-optimal solutions. The results through extensive 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 very challenging task due to unpredictable future motions of moving obstacles. The nonholonomic model and time-dependent collision checking impose highly nonlinear constraints on the planning task, making the problem even more intractable. The optimal solution is typically unachievable, and the planning completeness also cannot be ensured [5]. As a result, 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] under 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 some searching-based algorithms [9]–[12], which combines a long-horizon path searcher for improving the global optimality and a local collision avoidance strategy to ensure safety. 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 the motions of obstacles, which significantly limits the planning speed. Meanwhile, the differential constraints will reduce the volume of admissible state space, making the problem even more difficult.

To address these problems, we propose an online partial motion planner to generate dynamically feasible trajectories in highly dynamic environments. 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 will generate 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 method [5]). The algorithm performs velocity planning on the entire motion primitive and can provide a strict safety guarantee under the velocity extrapolation model.

The contributions of this work are as follows:

- We propose an efficient 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 searching and collision checking algorithms, we implement an online trajectory planning framework with very high planning frequency and success rates. The code will also be available soon.

* The corresponding author of this paper.

¹ The authors are with the Department of Electronic Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong SAR, 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, Hong Kong (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.

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 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 continuousness of the trajectory is not ensured. This problem is then solved by Luigi *et al.* [14] using 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 all designed for static environments but share a similar idea with ours.

State lattice is another discretization method that transforms the continuous state space into searching graphs. The state lattice is introduced by [17] for motion planning in static environments. The edges that connect adjacent states in state lattice are another type of motion primitive. Matthew *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 can be regarded as 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 based on an online collision checking process. 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 Fields based method [21]. Both 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 will be intractable.

III. PROBLEM DEFINITION

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

The motion model \mathcal{R} of a mobile robot is defined by its state function with differential constraints,

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

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

The map representation \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, $\mathcal{O}_i(t), i \in \{1, \dots, M\}$, the free space is further restricted, which imposes safety constraints on the planning,

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

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 lies in 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 typically 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. This imposes the boundary value constraints on the planning problem,

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

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,

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

The less control efforts are applied, the less state changes are obtained, and thus the generated trajectory is smoother.

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

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

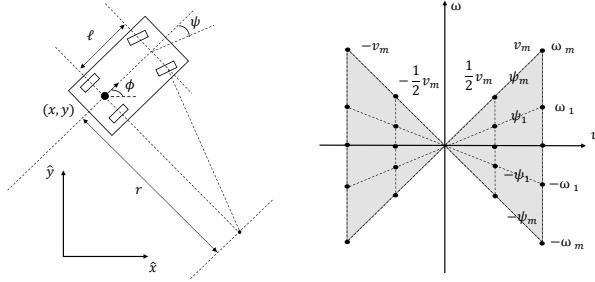
The solution for this problem defines a partial trajectory within the planning horizon T . For the remaining part that lies in $(T, T_g]$, the safety constraint is not required to be strictly satisfied. This is different from the conventional motion planning problem and is referred to as the PMP. Optimizing Eq. (5) in continuous space is intractable. We thus approximate the solution with motion primitives at the expense of optimality.

IV. GRAPH SEARCHING WITH MOTION PRIMITIVES

A. Robot Motion Model

There are many types of motion models for wheeled robots, including holonomic and nonholonomic ones. In this work, as shown in Fig. 1a, we adopt the canonical simplified car kinematics to model the robot,

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



(a) The simplified car kinematics. (b) The (v, ω) control set.
Fig. 1: The motion and control models for the wheeled robot.

with control transformations

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

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 transformed 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 angle ψ (or the minimum turning radius) of the robot.

In practice, a 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 need to make sure the control inputs along the trajectory always lie in the control set, i.e., $\mathbf{u}(t) \in \mathcal{U}$.

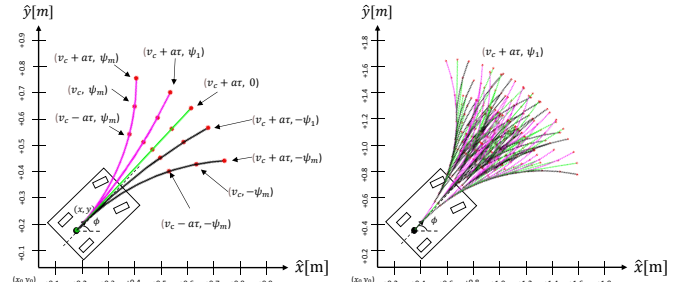
B. Motion Primitives

Motion primitive ξ_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 generated as follows,

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

where $(\tan \psi_m) / \ell = 1/r$ and r is the turning radius (Fig. 1a) of the primitive. It is worth noting that the linear velocity sometimes 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 angle, and each red point in the arch is a terminal state of the motion primitive generated with a sampled linear acceleration. The



(a) One-layer motion primitives. (b) Motion primitive tree.
Fig. 2: The generation of motion primitives based on a car model.

Algorithm 1 GeneratePrimitives($s_k, \mathcal{X}_g, \mathcal{O}$)

- 1: $\text{SUCC} \leftarrow \emptyset$
 - 2: **for** $\mathbf{u} \in \mathcal{U}_N$ **do**
 - 3: $\xi_k, \mathbf{s}_{k+1} \leftarrow \text{MOTIONPRIMITIVE}(\mathbf{s}_k, \mathbf{u}, \tau)$
 - 4: **if** $\text{ISVALID}(\mathbf{s}_{k+1}) \wedge \text{COLLISIONFREE}(\xi_k, \mathcal{O}, \tau)$ **then**
 - 5: $\mathbf{g}(\mathbf{s}_{k+1}) \leftarrow \mathbf{g}(\mathbf{s}_k) + \text{PRIMITIVECOST}(\xi_k)$
 - 6: $\mathbf{h}(\mathbf{s}_{k+1}) \leftarrow \text{HEURISTIC}(\mathbf{s}_{k+1}, \mathcal{X}_g)$
 - 7: $\mathbf{f}(\mathbf{s}_{k+1}) \leftarrow \mathbf{g}(\mathbf{s}_{k+1}) + \mathbf{h}(\mathbf{s}_{k+1})$
 - 8: $\mathbf{s}_{k+1}.\text{pre} \leftarrow \mathbf{s}_k$
 - 9: $\text{INSERT}(\mathbf{s}_{k+1}, \text{SUCC})$
 - 10: **end if**
 - 11: **end for**
 - 12: **return** SUCC
-

transformed control inputs along the motion primitive are constrained on a line segment within the control set, as shown in Fig. 1b. 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 structure and named primitive tree.

C. Online Graph Searching

Motion primitive ξ_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), and the cost of each primitive is $\|\mathbf{u}(t)\|^2 + \beta\tau$, where $\|\mathbf{u}(t)\|^2 = p_1 a^2 + p_2 \psi^2$. For the primitive with control $(a = 0, \psi = 0)$, $p_1 = 1, p_2 = 1$. Otherwise, p_1 and p_2 is set to > 1 . As a result, the planner prefers to approach the goal with maximum velocity in a straight line. By leveraging motion primitives, the PMP defined in Eq. (5) is transformed into a combinatorial optimization problem over the discretized space,

$$\begin{aligned} \min_{\xi_{0:K}, K} & \left(\sum_{k=0}^K \|\mathbf{u}_k\|^2 + \beta(K+1) \right) \tau \\ \xi_k((k+1)\tau) &= \xi_{k+1}((k+1)\tau), \quad \forall \xi_i \in \Xi_i, \\ \xi_k(t) &\notin \mathcal{M}_f \cap \mathcal{O}(t), \quad t \in [k\tau, (k+1)\tau], \quad \forall k \in \{0, \dots, \lceil T/\tau \rceil\}, \\ \xi_0(0) &= \mathbf{x}_0, \quad \xi_K \cap \mathcal{X}_g \neq \emptyset, \quad \xi_{\lceil T/\tau \rceil} \cap \mathcal{X}_{ICS} = \emptyset, \end{aligned} \quad (9)$$

where Ξ_i is the set of motion primitives generated for time duration $[i\tau, (i+1)\tau]$ with a depth of i in the primitive tree.

Algorithm 2 GraphSearching($s_0, \mathcal{X}_g, \mathcal{O}, \tau, T$)

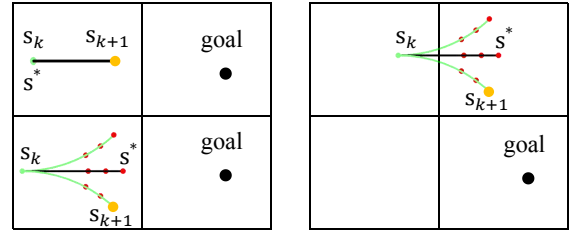
```

1: HMAP  $\leftarrow \emptyset$ , OPEN  $\leftarrow \emptyset$ 
2:  $s_0$ .key  $\leftarrow$  GENKEY( $s_0$ ),  $s_0$ .idx  $\leftarrow 0$ 
3:  $s_0$ .state  $\leftarrow \mathbf{x}_0$ ,  $g(s_0) \leftarrow 0$ 
4: HMAP.INSERT( $s_0$ ), OPEN.INSERT( $s_0$ )
5: while OPEN is not empty do
6:    $s_k \leftarrow$  OPEN.POP()
7:    $s_k$ .closed  $\leftarrow$  true
8:    $s_g \leftarrow$  GOALREACHED( $s, \mathcal{X}_g$ )
9:    $s_t \leftarrow$  TIMEREACHED( $s, T$ )
10:  /* the goal or planning horizon is reached */
11:  if  $s_g$ .state  $\in \mathcal{X}_g$  then
12:    return BACKTRACK( $s_g$ )
13:  else if  $s_t$ .state  $\notin \mathcal{X}_{ICS}$  then
14:    return BACKTRACK( $s_t$ )
15:  end if
16:  /* generate successors using motion primitives */
17:  SUCC  $\leftarrow$  GeneratePrimitives( $s_k, \mathcal{X}_g, \mathcal{O}$ )
18:  for  $s_{k+1} \in$  SUCC do
19:     $s_{k+1}$ .key  $\leftarrow$  GENKEY( $s_{k+1}$ ),  $s_{k+1}$ .idx  $\leftarrow 0$ 
20:     $s_{k+1}$ .state  $\leftarrow \mathbf{x}_{k+1}$ 
21:    elements  $\leftarrow$  HMAP.QUERY( $s_{k+1}$ .key)
22:    /* already reached by other primitives */
23:    if elements  $\neq \emptyset$  then
24:       $n \leftarrow$  LENGTH(elements)
25:       $s^* \leftarrow$  elements[ $n - 1$ ]
26:      case1  $\leftarrow s^*$ .key =  $s$ .key
27:      case2  $\leftarrow s^*$ .pre =  $s_{k+1}$ .pre
28:      case3  $\leftarrow \sim(\text{case1} \vee \text{case2})$ 
29:      if (case1  $\vee$  case2)  $\wedge$  ( $h(s_{k+1}) < h(s^*)$ ) then
30:         $s_{k+1}$ .idx  $\leftarrow n$ 
31:        HMAP.INSERT( $s_{k+1}$ ), OPEN.INSERT( $s_{k+1}$ )
32:      end if
33:      if case3  $\wedge s^*$ .closed == false  $\wedge g(s_{k+1}) < g(s^*)$  then
34:         $s_{k+1}$ .idx  $\leftarrow s^*$ .idx
35:        HMAP.REPLACE( $s^*, s_{k+1}$ )
36:      end if
37:      /* never reached by other primitives */
38:    else
39:      HMAP.INSERT( $s_{k+1}$ ), OPEN.INSERT( $s_{k+1}$ )
40:    end if
41:  end for
42: end while
43: return FAILURE

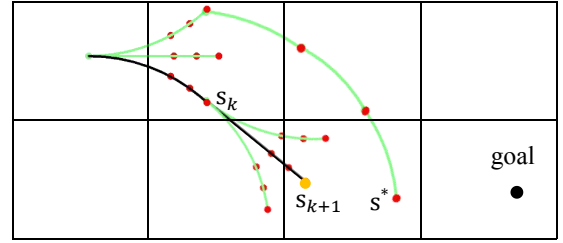
```

Based on such a discretization scheme, the planning problem becomes finding a sequence of collision-free motion primitives that connect the start and goal within the planning horizon while maintaining an ICS for the local goal.

This problem is typically solved by tree searching techniques, e.g., DFS and BFS, which however are not suitable for online trajectory generation due to their inefficiency. In this work, we propose to additionally discretize the state space for



(a) case1: s_k and s_{k+1} are in the same grid cell, and s^* is the parent or sibling of s_{k+1} . (b) case2: s_k and s_{k+1} connect two grid cells, and s^* is the sibling of s_{k+1} .



(c) case3: s_k and s_{k+1} connect two grid cells, and s^* is not the parent or sibling of s_{k+1} .

Fig. 3: Illustration of the three cases of node expansion. s^* is the optimal representation of the current grid cell. s_k is the parent node and 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.

aggregating the motion primitives to enable online trajectory generation in highly dynamic environments. The details are presented in Alg. 1 and 2, which are used to generate motion primitives and perform graph searching, respectively.

The graph is defined as $\mathcal{G}(\mathcal{X}, \mathcal{U})$, where each vertex is a discretized grid cell corresponding to a robot state $\mathbf{x} \in \mathcal{X}$, and each edge is a motion primitive generated by the control input $\mathbf{u} \in \mathcal{U}_N \subset \mathcal{U}$. In Fig. 3 (the ϕ dimension is not visualized), we demonstrate some cases of node expansion during graph building. It can be seen that there may exist multiple edges that connect two adjacent grid cells, and a grid cell may contain multiple state nodes. To make each grid being a vertex, we use the current optimal node s^* to represent that grid. In this way, the primitive tree is aggregated into a graph.

In Alg. 1, we present the generation of collision-free motion primitives for expanding state nodes (see Section IV-D), and the calculation of primitive cost and heuristic cost (see Section IV-E). Based on this algorithm and the definition of $\mathcal{G}(\mathcal{X}, \mathcal{U})$, we then perform online graph building and path searching with Alg. 2, which is a variant of A* algorithm. The HMAP is a hash map used for storing state nodes s . The map index of each node includes a hash key s .key and a primitive index s .idx (line 2), which are used to locate the grid cell that s belongs to and the position of s within the grid cell, respectively. In addition to the index field, s also records the robot state \mathbf{x} , i.e., s .state, the current cost value (line 3), and some other auxiliary fields. In lines 8-15, we check whether the goal or planning horizon is achieved, and make sure the local goal s_t is ICS free (see next section). Lines 18-42 present the node expansion and trajectory generation process. If a grid

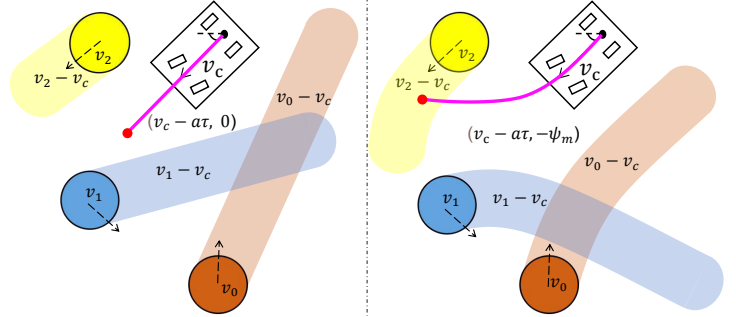
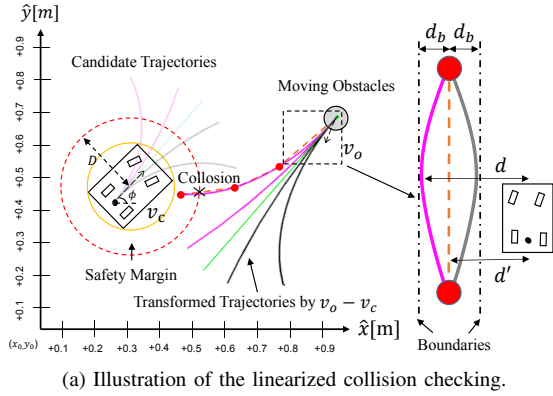


Fig. 4: Collision and ICS checking during graph searching.

cell is never reached, we just expand a new state node for this cell (lines 38-40); Otherwise, we need to identify the optimal representation for this cell (lines 22-36). In Fig. 3, 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 challenging and time-consuming problems in dynamic motion planning. The typical type of method is to discretize the target trajectory and then check collisions for each discretized waypoint by a forward simulation process. However, 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 ξ_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 ξ_k will be ensured.

To achieve this point, as shown in Fig. 4a, we first transform the obstacle trajectory by subtracting ξ_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. 4a) and then connect these waypoints as line segments to linearize the transformed trajectory. As shown in the right side of Fig. 4a, 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.

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. 4b, 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.

E. Heuristic Design

The length of Reed-Shepp curves without considering moving obstacles is adopted as the heuristic (line 6 in Alg. 1). These curves are dynamically feasible for car kinematics model and have the shortest paths between the current state \mathbf{s} and a given goal state \mathbf{s}_g . Reeds-Shepp heuristic is admissible in our problem, since it never overestimates the actual cost, i.e., $0 \leq h(\mathbf{s}, \mathbf{s}_g) \leq g(\mathbf{s}, \mathbf{s}_g)$. This heuristic is also consistent, which means that $h(\mathbf{s}_a, \mathbf{s}_c) \leq g(\mathbf{s}_a, \mathbf{s}_b) + h(\mathbf{s}_b, \mathbf{s}_c)$ holds for any \mathbf{s}_a and \mathbf{s}_b . This can be easily proved by leveraging the facts that $h(\mathbf{s}_a, \mathbf{s}_c) \leq h(\mathbf{s}_a, \mathbf{s}_b) + h(\mathbf{s}_b, \mathbf{s}_c)$ and $0 \leq h(\mathbf{s}_a, \mathbf{s}_b) \leq g(\mathbf{s}_a, \mathbf{s}_b)$. In dynamic motion planning, the Reed-Shepp heuristic actually underestimates the actual cost too much, since the moving obstacles will significantly increase the path length. There exists a scale between the Reed-Shepp heuristic and the actual cost, i.e., $g(\mathbf{s}, \mathbf{s}_g) = \alpha h(\mathbf{s}, \mathbf{s}_g)$, $\alpha > 1$. A proper scale α on the heuristic can make it more informative, and enable the algorithm to converge quickly towards the solution. However, α depends on the future motions of moving obstacles and cannot be calculated at the current stage. In this work, we adopt an empirical value $\alpha = 1.3$, which can significantly improve the planning efficiency. The deficiency is that the solution given by Alg. 2 cannot be ensured optimal, since the scaled heuristic may become inadmissible sometimes.

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.

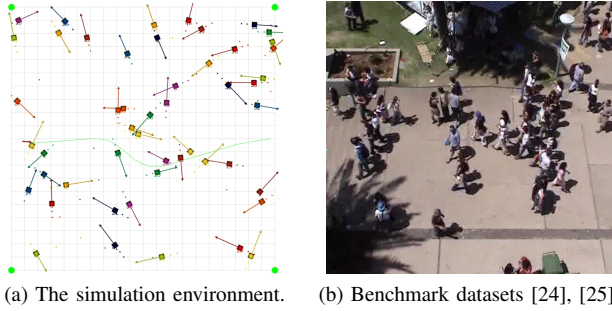


Fig. 5: The experiment environments and datasets used in this work.

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 will be 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 variance on speed can help test the robustness of different collision checking strategies.

The default parameter settings for the simulation 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 of the experiment, one of the parameters is changed and the remaining is set to default. The middle points on the left and right sides of the environment, shown in Fig. 5a, are taken as planning start and goal, respectively. Each test repeats 30 times, and the average success rate and time cost 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. 6.

We can see that all planners exhibit a worse performance when the number of moving agents or the safe margin is increased, shown in Fig. 6a and 6b. The reason is obvious that the feasible region in the planning space is occupied by the 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. 6a, while maintaining a competitive performance in Fig. 6b. Such a result is essentially caused by the different collision checking strategies. The DC method performs path searching on a sparse triangle graph generated by the Delaunay triangulation algorithm. The collision checking is limited to safely passing through a gate defined by the common edge of the current triangle with the adjacent one, which however cannot provide safety guarantee, especially in highly dynamic environments. The VO method considers all the surrounding obstacles for collision checking, thus exhibiting a better per-

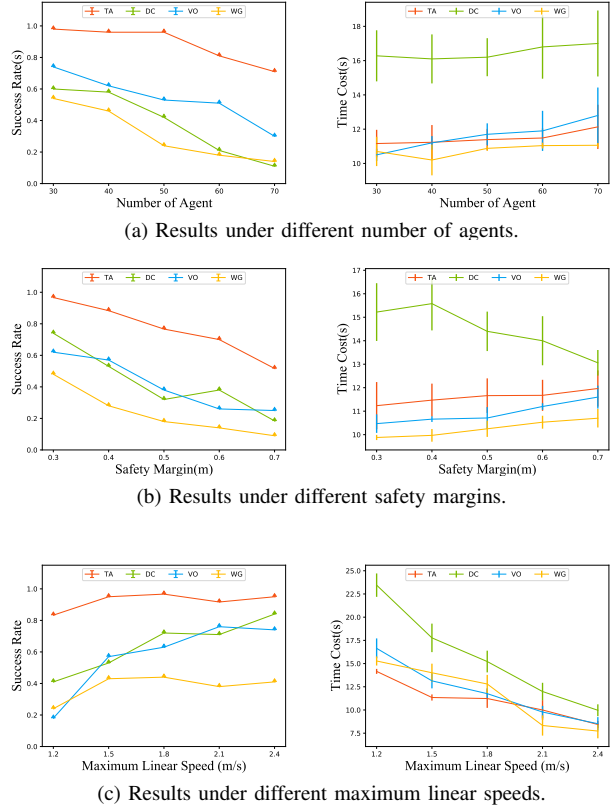


Fig. 6: Performance on the simulation environment.

formance 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. 6c, we can see 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 fast navigation through the edge of triangles under smaller topology changes of the graph, hence the success rate can be improved. For the TA and VO method, on the one hand, higher speeds increase the reactive ability of the robot and thus the success rate. On the other hand, higher speeds increase the horizon of collision checking, which is not beneficial for fast-planning. The success rate is thus affected to some extent, as shown in Fig. 6c.

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 over the baseline methods.

B. Experiment on Benchmark Datasets

We then evaluate our method on three public publicly available datasets from ETH [30] and UCY [31]. There are totally seven sequences of videos captured from different scenarios, i.e., *stu001*, *stu003*, *zara01*, *zara02*, *zara03*, *biwi_eth*, and *biwi_hotel*. All the sequences are interpolated and played at a frequency of 10Hz. As demonstrated in Fig. 7, the *stu*, *zara*,

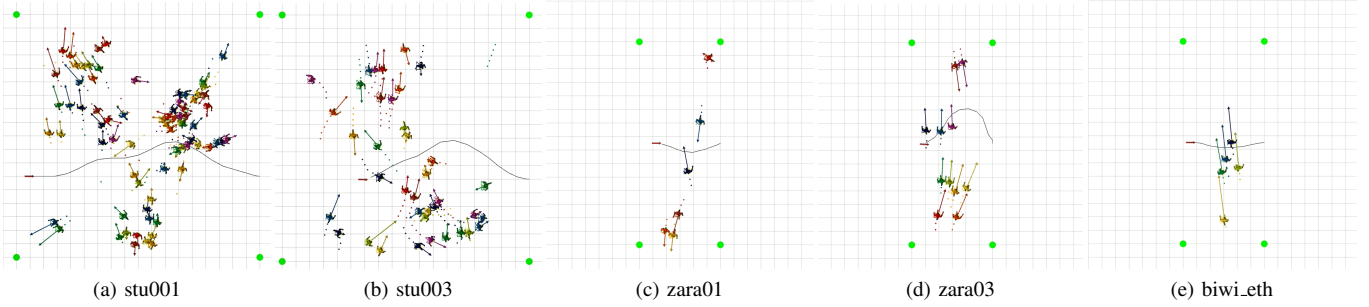


Fig. 7: The planning results of our online state-time planner on different data sequences.

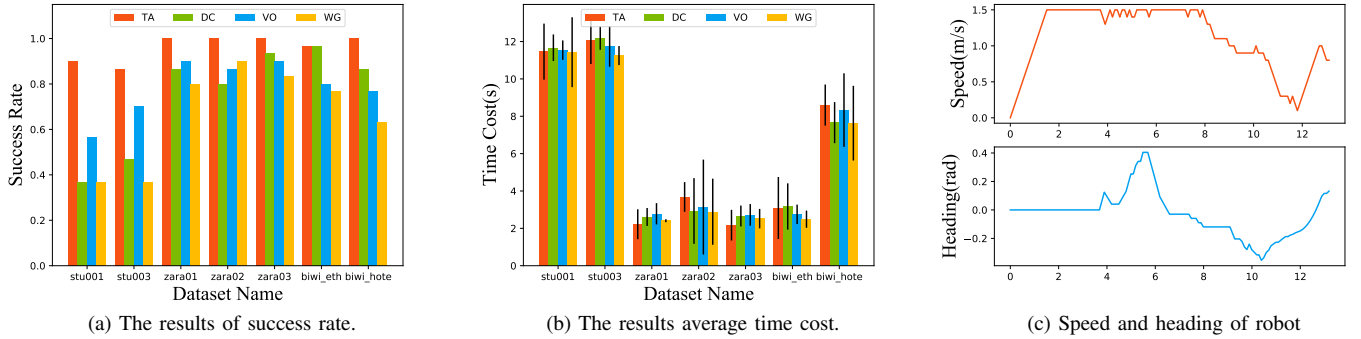


Fig. 8: Performance on the benchmark datasets.

and *biwi* sequences are with a high, medium, and low crowd density, respectively. During experiments, the middle points on the left and right sides of the environment, shown in Fig. 7, are taken as planning start and goal, respectively. The maximum linear speed is set to 1.5m/s. The safe distance is set to 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 experiment results are presented in Fig. 8.

In this part, we will analyze the success rate and time cost at the same time. WG achieves the lowest time cost in all datasets with a low success rate in *stu001* and *stu003*, as shown in 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 straight-line strategy has a large chance to collide in the environment with high crowd density, which results in a low success rate. As VO avoids the obstacle aggressively without considering the future motion of pedestrians, 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 the dense situation which contributes to a large average 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 high penalty factor is given to deceleration. We still claim that TA shows advantages over other planners. Because TA achieves the highest success rate in all the datasets and maintains a relatively low time cost. In general, we can

observe a similar phenomenon in the simulation environment. As shown in Fig. 8a, both TA and VO methods achieve a higher success rate in dense *stu* sequences, while DC performs better than VO in sparse *biwi* sequences. Fig. 8c shows the speed-time graph and heading-time graph of a sample trajectory generated by TA planner in dataset *stu001*. We can observe that the speed and heading are smooth at the beginning, as TA prefers to go in a straight line without deceleration. The rapid changes in linear speed are caused by unexpected collision avoidance, as the constant velocity assumption is inaccurate most of the time. The computation time of TA in different benchmark datasets is shown in Table I. In the experiment, we use 9 motion primitives for tree expansion and the planning horizon for each step is 0.5s. The coefficient for heuristic is $\alpha = 1.3$. These are the key parameters related to the computation time. We can see that the proposed planner has a very high efficiency.

TABLE I: Computation time on benchmark dataset

Scene	stu001	stu003	zara01	zara02	biwi.eth	biwi_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 to generate dynamically feasible trajectories for car-like robots in highly dynamic environments. The primitives generated by discretized control is over densely distributed in

state-space, which is the major disadvantage 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 which linearizes relative motions between the robot and obstacles and then checks collisions by comparing a point-line distance. Benefiting from the efficient state representation and collision checking methods, the planner can generate feasible trajectories within milliseconds while maintaining a high success rate in highly dynamic environments. Extensive experiments also demonstrate its significant advantages. Our proposed method facilitates the trajectory planning for car-like robots based on control space discretization method in the highly dynamic environment. The major limitation of the algorithm lies in the un-informative heuristic functions, which are also the focus of our next-step work.

REFERENCES

- [1] D. Zhu, Y. Du, Y. Lin, H. Li, C. Wang, X. Xu, and M. Q.-H. Meng, "Hawkeye: Open source framework for field surveillance," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6083–6090.
- [2] D. Zhu, T. Li, D. Ho, and M. Q.-H. Meng, "Deep reinforcement learning supervised autonomous exploration in office environments," in *Robotics and Automation (ICRA), 2018 IEEE International Conference on*. IEEE, 2018.
- [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*. IEEE, 2005, pp. 2210–2215.
- [7] T. Fraichard and H. Asama, "Inevitable collision states? step towards safer robots?" *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.
- [8] H.-T. 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)*. IEEE, 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 *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 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*, 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*, 2009, pp. 1662–1668.
- [12] M. Ruffli 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 *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 473–479 vol.1.
- [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: Pre-computed 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 *IEEE International Conference on Robotics and Automation*, 2005.
- [22] D. Wilkie, J. Van Den Berg, and D. Manocha, "Generalized velocity obstacles," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 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.